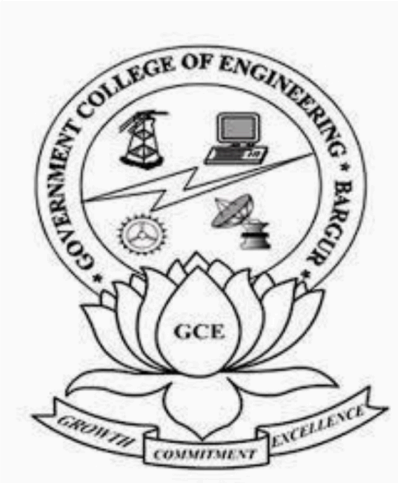
GOVERNMENT COLLEGE OF ENGINEERING, BARGUR

KRISHNAGIRI – 635 104

(An Autonomous Institution Affiliated to Anna University – Chennai)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECEMBER 2022

20SPC310 - DATA STRUCTURES LABORATORY

NAME: _____

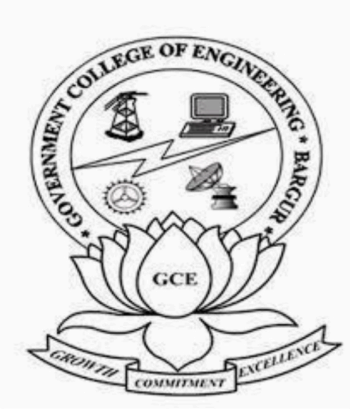
CLASS: _____

SEMESTER: _____

REGISTER NUMBER: _____

GOVERNMENT COLLEGE OF ENGINEERING BARGUR, KRISHNAGIRI – 635 104

(An Autonomous Institution Affiliated to Anna University – Chennai)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REGISTER NUMBER :

SUBJECT CODE : 20SPC310

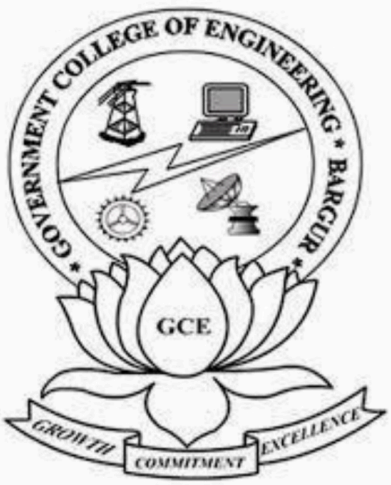
SUBJECT : DATA STRUCTURES LABORATORY

Certified that this is the bonafide record of work done by the student_____ of _____ semester of B.E computer science engineering (Full-time) branch during the academic year 2021-2022.

FACULTY IN-CHARGE **HEAD OF THE DEPARTMENT**

Submitted for the Practical Examination held on_____ at Government College of Engineering, Bargur.

INTERNAL EXAMINER **EXTERNAL EXAMINER**



GOVERNMENT COLLEGE OF ENGINEERING, BARGUR

(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA)

VISION: To endow with exceptional computer science education by building well-built training and research environment

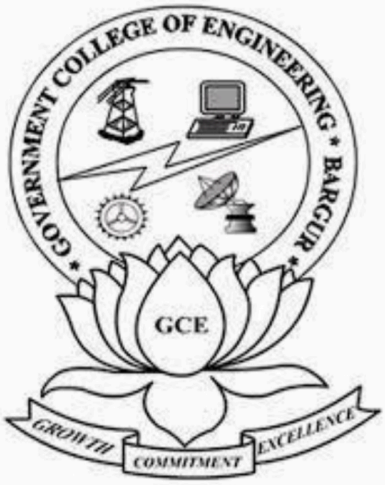
MISSION: To offer high quality graduate program in computer in computer science and to prepare students for professional career or higher studies, The department promotes brilliance in teaching, research, mutual activities and construction assistance to the public

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Be good at in professional career and/or higher education by acquiring understanding in mathematical, computing and engineering principles.

PEO 2: Explore real life problems, design computing systems apt to its solutions that are technically sound, economically practicable and socially satisfactory.

PEO 3: Demonstrate professionalism, ethical attitude, communication skills, team work in their profession and adapt to current trends by engaging in lifelong learning.



GOVERNMENT COLLEGE OF ENGINEERING, BARGUR

(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA)

VISION: To endow with exceptional computer science education by building well-built training and research environment

MISSION: To offer high quality graduate program in computer in computer science and to prepare students for professional career or higher studies, The department promotes brilliance in teaching, research, mutual activities and construction assistance to the public

20SPC310 - DATA STRUCTURES LABORATORY

COURSE OBJECTIVES

COBJ1: To implement linear and non-linear data structures.

COBJ2: To understand the different operations of search trees and graph traversal algorithms.

COBJ3: To get familiarized to sorting, searching algorithms and hashing techniques.

COURSE OUTCOMES

CO1: Apply generic programming technique to implement any data structure

CO2: Identify appropriate search trees for an application.

CO3: Make use of graphs in problem solving.

CO4: Develop the various sorting algorithms and compare them.

CO5: Create a program for hash applications.

.

INDEX

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	SIGNATURE OF FACULTY

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	SIGNATURE OF FACULTY

EX.NO : 1A	ARRAY IMPLEMENTATION OF LIST ADT
DATE:	

AIM:

To write a C Program for implementation of list using array.

ALGORITHM:

Step 1 - Start

Step 2 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 3 - Declare all the functions used in list implementation.

Step 4 - Create a one dimensional array with fixed size (intL[SIZE])

Step 5 - In main method, display menu with list of operations and make suitable function calls to perform operation using switch case depends on the users choice.

Step 6 - Stop

CREATE()

Step 1 - Start

Step 2 - Read the elements to be inserted using for loop and assign L[i]=x.

Step 3 - Stop

INSERT()

Step 1 - Start

Step 2 - Increment size by 1.

Step 3 - Read the position of the element to be inserted.

Step 3.1 - If (p>=size), then display “Position is out of range”.

Step 3.2 – Else insert the element at the specified using for loop

Step 4 - Stop

DELETE()

Step 1 - Start

Step 2 - Read the position of the element to be deleted.

Step 2.1 - If (p<0 or p>=size), then display “Position is out of range”.

Step 2.2 – Else display and delete the element.

Step 3 – Stop

FIND()

Step 1 - Start

Step 2 - Read a value X ,then define a variable 'i' and initialize with 0. Check if X==L[i]

Step 3.1 - If yes, display “ELEMENT FOUND !!!” , break.

Step 3.2 – If no, display “ELEMENT NOT FOUND !!!”.

Step 4 - Stop

DISPLAY()

Step 1 - Start

Step 2 - Check whether list is EMPTY. (stack<0)

Step 3 - If it is EMPTY, then display "List is EMPTY!!!" and terminate the function.

Step 4- If it is NOT EMPTY, then define a variable 'i' and initialize with 0. Display L[i] value and increment i value by one (i++).

Step 5 - Repeat above step until i reaches size.

Step 6 – Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void create();
void insert();
```

```
void Delete();
void display();
void find();
int x,size,i,ch,n,p;
int L[50];
void main()

{
   printf("\n\t*****IMPLEMENTATION OF LIST USING ARRAY*****");
   printf("\n\nEnter the size of list:");
   scanf("%d",&size);
   while(1)
   {
      printf("\nMenu:\n\t1.create\n\t2.insert\n\t3.Delete\n\t4.display\n\t5.find\n\t6.exit");
      printf("\nEnter your choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         create();
         break;
      case 2:
         insert();
         break;
      case 3:
         Delete();
         break;
      case 4:
         display();
         break;
      case 5:
         find();
         break;
      case 6:
         exit(0);
      default:
         printf("Try Again");
         break;
      }
   }
}
void create()
{
   printf("\nEnter the element to be inserted:\n");
   for(i=0;i<size;i++)
```

```
    {
        scanf("%d",&L[i]);
    }
}


void insert()
{
    size=size+1;
    printf("enter the position to be inserted:\n");
    scanf("%d",&p);
    if(p>size)
        printf("Position is out of range");
    else
    {
        for(i=size-2;i>=p;i--)
        {
            L[i+1]=L[i];
        }
        printf("enter the element to be inserted:");
        scanf("%d",&x);
        L[i+1]=x;
    }
}
void Delete()
{
    printf("enter the position to be deleted:\n");
    scanf("%d",&p);
    if((p<0)||(p>=size))
        printf("Position is out of range");
    else
    {
        printf(“Deleted Element:%d”,L[p]);
        for(i=p+1;i<size;i++)
        {
            L[i-1]=L[i];
        }
        L[i-1]=0;
        size=size-1;
    }
}
void display()
{
    if(size<0)
        printf("List is empty");
```

```
        else
        {
           for(i=0;i<size;i++)
           {

           printf("\t%d",L[i]);
           }
        }
    }
    void find()
    {
       printf("\nEnter the element to be found:");
       scanf("\t%d",&x);
       for(i=0;i<size;i++)
       {
         if(L[i]==x)
         {
            printf("Element is found in %d th position",i);
            break;
         }
         if(i==size)
         {
          printf("Element not found");
         }
       }
    }
```

OUTPUT:

```
        *****IMPLEMENTATION OF LIST USING ARRAY*****

    Enter the size of stack:3

    MENU:-
    1.create
    2.insert
    3.Delete
    4.display
    5.find
    6.exit
    Enter your choice:1

    Enter the Element to be inserted:
    2
    4
```

6

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:4
2 4 6

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:5

Enter the Element to be found:3
Element is not Found

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:5

Enter the Element to be found:6
Element is Found at 2th position

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:2

```
Enter the position to be inserted:1
Enter the Element to be inserted:8

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:4
     2        8        4        6

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:2

Enter the position to be deleted:6
Position is out of range

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:2

Enter the position to be deleted:3
Deleted element is:6

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:4
```

2 8 4

MENU:-
1.create
2.insert
3.Delete
4.display
5.find
6.exit
Enter your choice:6

RESULT:

Thus the C program to implement list using array has been executed and output is verified successfully**.**

EX.NO : 1B	ARRAY IMPLEMENTATION OF STACK ADT
DATE :	

AIM:

To write a C Program for implementation of stack using array.

ALGORITHM:

Step 1 - Start

Step 2 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2 - Declare all the functions used in stack implementation.

Step 3 - Create a one dimensional array with fixed size (int stack[SIZE])

Step 4 - Define a integer variable 'top' and initialize with '-1'. (int top = -1)

Step 5 - In main method, display menu with list of operations and make suitable function calls to perform operation using switch case depends on the users choice.

Step 6 - Stop

PUSH()

Step 1 - Check whether stack is FULL. (top == SIZE-1)

Step 2 - If it is FULL, then display "Stack is FULL!!!" and terminate the function.

Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

POP()

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

PEEP()

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, read a value X ,then define a variable 'i' and initialize with top. Check if X==stack[i]

Step 3.1 - If yes, display “ELEMENT FOUND !!!” , break.

Step 4 - Decrement i value by one (i--).

Step 5 - Repeat above step until i value becomes '0'.

Step 6 – Check whether (i==-1) , if yes display “ELEMENT NOT FOUND !!!”

DISPLAY()

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).

Step 4 - Repeat above step until i value becomes '0'

ISFULL()

Step 1 - Check whether stack is FULL. (top == SIZE-1)

Step 2 – If yes, display "Stack is FULL !!!" , else display "Stack is not FULL!!!"

ISEMPTY()

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If yes, display "Stack is EMPTY!!!", else display "Stack is not EMPTY!!!"

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void push();
void pop();
void peep();
void IsFull();
void IsEmpty();
void display();
```

```
int a,b,c,n,ch,size,i;
int top=-1;
int stack[50];
void main()
{
   printf("\n\t*****IMPLEMENTATION OF STACK USING ARRAY*****");
   printf("\n\nEnter the size of stack:");
   scanf("%d",&size);
   while(1)
   {
      printf("\nMENU:-\n\t1.push\n\t2.pop\n\t3.peep\n\t4.display\n\t5.IsFull\n\t6.IsEmpty\n\t7.exit");
      printf("\nEnter your choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         push();
         break;
      case 2:
         pop();
         break;
      case 3:
         peep();
         break;
      case 4:
         display();
         break;
      case 5:
         IsFull();
         break;
      case 6:
         IsEmpty();
         break;
      case 7:
         exit(0);
      default:
         printf("Try again");
         break;
      }
   }
}
void push()
{
 if(top==size-1)
```

```
   printf("\nStack is Full");
 else
 {
   printf("\nEnter the Element to be inserted:");
   scanf("%d",&a);
   top=top+1;
   stack[top]=a;
 }
}
void pop()
{
 if(top==-1)
   printf("\nStack is Empty");
 else
 {
   b=stack[top];
   printf("\nDeleted element is:%d",b);
   top=top-1;
 }
}
void peep()
{
 if(top==-1)
   printf("\nStack is Empty");
 else
 {
   printf("\nEnter the element to be searched:");
   scanf("%d",&c);
   for(i=top;i>=0;i--)
   {
    if(stack[i]==c)
    {
     printf("\nElement is Found");
     break;
    }
   }
   if(i==-1)
     printf("\nElement is not Found");
 }
}
void IsFull()
{
 if(top==size-1)
   printf("Stack is Full");
```

```
 else
   printf("Stack is not Full");
}
void IsEmpty()
{
 if(top==-1)
   printf("Stack is Empty");
 else
   printf("Stack is not Empty");
}
void display()
{
 if(top==-1)
   printf("\nStack is Empty");
 else
 {
   for(i=top;i>=0;i--)
   {
     printf("\t%d",stack[i]);
   }
 }
}
```

OUTPUT:

```
        *****IMPLEMENTATION OF STACK USING ARRAY*****

Enter the size of stack:4
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:1
Enter the Element to be inserted:2
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
```

```
        6.IsEmpty
        7.exit
Enter your choice:1
Enter the Element to be inserted:4
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:1
Enter the Element to be inserted:6
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:4
        6       4       2
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:3
Enter the element to be searched:4
Element is Found
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:3
```

```
Enter the element to be searched:10
Element is not Found

MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:2
Deleted element is:6
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:4
        4       2
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:5
Stack is not Full
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:2
Deleted element is:4
```

```
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:2

Deleted element is:2
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:4
Stack is Empty
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:6
Stack is Empty
MENU:-
        1.push
        2.pop
        3.peep
        4.display
        5.IsFull
        6.IsEmpty
        7.exit
Enter your choice:7
```

RESULT:

Thus the C program for implementation of stack using array was written successfully and output is verified.

EX.NO : 1C	ARRAY IMPLEMENTATION OF QUEUE ADT
DATE :	

AIM:

To write a C Program for implementation of queue using array.

ALGORITHM:

Step 1 - Start

Step 2 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 3 - Declare all the functions used in queue implementation.

Step 4 - Create a one dimensional array with fixed size (int Queue[SIZE])

Step 5 - Define a integer variable 'FRONT' , 'REAR' and initialize with '-1'. (int FRONT=-1, REAR=-1)

Step 6 - In main method, read size of Queue, display menu with list of operations and make suitable function calls to perform operation using switch case depends on the users choice.

Step 7 - stop

ENQUEUE()

Step 1 - Check whether queue is FULL. (REAR == SIZE-1)

Step 2 - If it is FULL, then display "Queue is FULL!!!" and terminate the function.

Step 3 - If it is NOT FULL, read a value then check if (FRONT==-1)&&(REAR==-1), if yes, set FRONT=0, REAR=0 else increment REAR value by one (REAR++) and set Queue[REAR] to value (Queue[REAR] = value).

DEQUEUE()

Step 1 - Check whether queue is EMPTY. (FRONT == -1)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then delete Queue[FRONT] check if (FRONT==REAR), if yes, set FRONT=-1, REAR=-1 else increment FRONT value by one (FRONT++).

PEEP()

Step 1 - Check whether queue is EMPTY. (FRONT == -1)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, read a value X ,then define a variable 'i' and initialize with FRONT. Check if X==Queue[i].

Step 3.1 - If yes, display “ELEMENT FOUND !!!” , break.

Step 4 - Increment i value by one (i++).

Step 5 - Repeat above step until i value becomes 'REAR'.

Step 6 – Check whether (i==REAR+1) , if yes display “ELEMENT NOT FOUND !!!”

DISPLAY()

Step 1 - Check whether queue is EMPTY. (FRONT == -1)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with FRONT. Display Queue[i] value and increment i value by one (i++).

Step 4 - Repeat above step until i value becomes ‘REAR’.

ISFULL()

Step 1 - Check whether queue is FULL. (REAR == SIZE-1)

Step 2 – If yes, display "Queue is FULL !!!" , else display "Queue is not FULL!!!"

ISEMPLTY()

Step 1 - Check whether queue is EMPTY. (FRONT == -1)

Step 2 - If yes, display "Queue is EMPTY!!!", else display "Queue is not EMPTY!!!"

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void Enqueue();
```

```
void Dequeue();
void Search();
void Display();
void IsQueueFull();
void IsQueueEmpty();
int a,b,c,n,ch,size,i,Queue[50];
int FRONT=-1,REAR=-1;
int main()
{
 printf("\n\t****IMPLEMENTATION OF QUEUE USING ARRAY****");
 printf("\n\nEnter the size of Queue:");
 scanf("%d",&size);
 while(1)
 {
  printf("\n\nMENU:\n\t1.Enqueue\n\t2.Dequeue\n\t3.Search\n\t4.Display\n\t5.IsQueueFull
\n\t6.IsQueueEmpty\n\t7.Exit");
  printf("\nEnter your choice:");
  scanf("%d",&ch);
  switch(ch)
  {
     case 1:
        Enqueue();
        break;
     case 2:
        Dequeue();
        break;
     case 3:
        Search();
        break;
     case 4:
        Display();
        break;
     case 5:
        IsQueueFull();
        break;
     case 6:
        IsQueueEmpty();
        break;
     case 7:
        return(0);
     default:
        printf("Try again");
        break;
  }
```

```
  }
}
void Enqueue()
{
 if(REAR==size-1)
   printf("Queue overflow");
 else
 {
   printf("\nEnter the element to be insert:");
   scanf("%d",&a);
   if((REAR==-1)&&(FRONT==-1))
   {
     REAR=0;
     FRONT=0;
   }
   else
     REAR=REAR+1;
   Queue[REAR]=a;
 }
}
void Dequeue()
{
 if(FRONT==-1)
   printf("Queue underflow");
 else
 {
   b=Queue[FRONT];
   printf("\nDeleted element is:%d",b);
   if(FRONT==REAR)
   {
     FRONT=-1;
     REAR=-1;
   }
   else
     FRONT=FRONT+1;
 }
}
void Search()
{
 if(FRONT==-1)
   printf("\nQueue is Empty");
 else
 {
   printf("\nEnter the element to be searched:");
```

```
        scanf("%d",&c);
        for(i=FRONT;i<=REAR;i++)
        {
         if(Queue[i]==c)
         {
          printf("\nElement is found at %d position",i);
          break;
         }
        }
        if(i==REAR+1)
        {
         printf("\nElement is not found");
        }
       }
      }
      void Display()
      {
       if(FRONT==-1)
        printf("\nQueue is Empty");
       else
       {
        for(i=FRONT;i<=REAR;i++)
        {
         printf("\t%d",Queue[i]);
        }
       }
      }
      void IsQueueFull()
      {
       if(REAR==size-1)
        printf("Queue is Full");
       else
        printf("Queue is not full");
      }
      void IsQueueEmpty()
      {
       if(FRONT==-1)
        printf("Queue is empty");
       else
        printf("Queue is not empty");
      }
```

OUTPUT:

****IMPLEMENTATION OF QUEUE USING ARRAY****

```
Enter the size of Queue:4

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:1

Enter the element to be insert:10

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:1

Enter the element to be insert:20

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:1

Enter the element to be insert:30

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
```

```
        6.IsQueueEmpty
        7.Exit
Enter your choice:4
        10      20      30

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:3

Enter the element to be searched:40

Element is not found

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:3

Enter the element to be searched:20

Element is found at 1 position

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:2

Deleted element is:10
```

```
MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:4
        20      30

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:5
Queue is not full

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:2

Deleted element is:20

MENU:-
        1.Enqueue
        2.Dequeue
        3.Search
        4.Display
        5.IsQueueFull
        6.IsQueueEmpty
        7.Exit
Enter your choice:2
```

Deleted element is:30

MENU:-
1.Enqueue
2.Dequeue
3.Search
4.Display
5.IsQueueFull
6.IsQueueEmpty
7.Exit
Enter your choice:4

Queue is Empty

MENU:-
1.Enqueue
2.Dequeue
3.Search
4.Display
5.IsQueueFull
6.IsQueueEmpty
7.Exit
Enter your choice:6
Queue is empty

MENU:-
1.Enqueue
2.Dequeue
3.Search
4.Display
5.IsQueueFull
6.IsQueueEmpty
7.Exit
Enter your choice:7

RESULT:

Thus the C program for implementing Queue ADTs using array was written and output is verified successfully.

EX.NO. : 2A	LINKED LIST IMPLEMENTATION OF STACK ADT
DATE:	

AIM:

To write a c program for linked list implementation of stack .

ALGORITHM:

step1: Start.

step2: Using while loop perform varies operations in linked list repeatedly until return (exit) statement is executed.
step3: Read choice.

step4: Switch case is used to select what operations to be performs and perform the operations according to the case.
step5: Stop.

PUSH ( )

step 1: Start

step 2: Create a newNode with given value.

step 3: Check whether stack is Empty (top == null)
step 4: If it is Empty, then set top=newNode.
step 5: If it is Not Empty, then set newNode → next = top.
step 6: Finally, set top = newNode..
step 7: Stop.

POP ( )

step 1:Start

step 2:Check whether stack is Empty (top == NULL).

step 3: If it is Empty, then display "Stack is Empty& Deletion is not possible" and terminate the function
step 4: If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
step 5: Then set 'top = temp → next'.
step 6: Finally, delete 'temp'.
step 7:Stop.

PEEP ( )

Step 1:Start.

Step 2:Set count=0

Step 3:Read element x to search

Step 4: traverse the list using while loop

Step 5:compare the element x with list elements

Step 6:If x==list elements print “element found” else print“element not found”
Step 7:Stop.

DISPLAY ( )

step 1: start.

step 2: Check whether top == null

step 3: If it is Empty, then display 'stack is Empty' and terminate the function.

step 4: If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
step 5: Keep displaying temp → data until temp reaches to the last node
step 6: Finally display temp → data
step 7: Stop.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define  null 0
void push();
void pop();
void peep();
void display();
int ch,item,b;
struct node
{
int data;
struct node*next;
}*top=null,*top,*New,*temp;
void main()
{
printf("\n\t**DYNAMIC IMPLEMENTATION OF STACK**");
while(1)
  {
   printf("\nMenu:\n\t1.push\n\t2.pop\n\t3.peep\n\t4.display\n\t5.exit");
   printf("\n Enter you choice");
   scanf("%d",&ch);
   switch(ch)
     {
      case 1:
       push();
        break;
       case 2:
        pop();
      break;
      case 3:
```

```
        peep();
        break;
        case 4:
        display();
        break;
        case 5:
        exit(0);
        default:
         printf("try again");
        }
    }
}
void push()
{
  New=(struct node*)malloc(sizeof(struct node));
       if(New==null)
          printf("\nout of space");
          else
          {
              printf("\nEnter the element to be insert:");
             scanf("%d",&item);
             New->data=item;
             New->next=top;
             top=New;
          }
}
void pop()
{
   if (top==null)
   printf("stack is empty");
   else
   {
      temp=top;
      top=temp->next;
      printf("deleted element is:%d",temp->data);
      free(temp);
   }
}
void peep()
{
   int count=0;
   if (top==null)
   printf("stack is empty");
```

```
   else
   {
      printf("\nEnter the element to search:");
      scanf("%d",&item);
      temp=top;
      do
      {
         count=count+1;
         if(item==temp->data)
           {
            printf("element found at position");
            printf("%d",count);
            break;
           }
         else
          temp=temp->next;
       }
         while(temp!=null);
if(temp==null)
printf("element not found");
   }}
void display()
{
  if (top==null)
  printf("stack is empty");
  else
    {
      temp=top;
      printf("\nThe elements in the list are");
      while(temp!=null)
        {
          printf("\n%d",temp->data);
          temp=temp->next;
        }}}
```

OUTPUT:

```
      **DYNAMIC IMPLEMENTATION OF STACK**
Menu:
      1.push
      2.pop
      3.peep
      4.display
      5.exit
 Enter you choice1
```

```
Enter the element to be insert:1
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice1
Enter the element to be insert:2
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice1
Enter the element to be insert:3
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice1
Enter the element to be insert:4
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice4
The elements in the list are
4
3
2
1
Menu:
1.push
2.pop
3.peep
 4.display
```

```
        5.exit
 Enter you choice2
deleted element is:4
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice4
The elements in the list are
3
2
1
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice3
Enter the element to search:4
element not found
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice3
Enter the element to search:2
element found at position2
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice5
```

RESULT:

Thus the C program to implement stack using dynamic implementation was written and output is verified successfully.

EX.NO. : 2A	LINKED LIST IMPLEMENTATION OF STACK ADT
DATE:	

AIM:

To write a c program for linked list implementation of stack .

ALGORITHM:

step1: Start.

step2: Using while loop perform varies operations in linked list repeatedly until return (exit) statement is executed.
step3: Read choice.

step4: Switch case is used to select what operations to be performs and perform the operations according to the case.
step5: Stop.

PUSH ( )

step 1: Start

step 2: Create a newNode with given value.

step 3: Check whether stack is Empty (top == null)
step 4: If it is Empty, then set top=newNode.
step 5: If it is Not Empty, then set newNode → next = top.
step 6: Finally, set top = newNode..
step 7: Stop.

POP ( )

step 1:Start

step 2:Check whether stack is Empty (top == NULL).

step 3: If it is Empty, then display "Stack is Empty& Deletion is not possible" and terminate the function
step 4: If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
step 5: Then set 'top = temp → next'.
step 6: Finally, delete 'temp'.
step 7:Stop.

PEEP ( )

Step 1:Start.

Step 2:Set count=0

Step 3:Read element x to search

Step 4: traverse the list using while loop

Step 5:compare the element x with list elements

Step 6:If x==list elements print “element found” else print“element not found”
Step 7:Stop.

DISPLAY ( )

step 1: start.

step 2: Check whether top == null

step 3: If it is Empty, then display 'stack is Empty' and terminate the function.

step 4: If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
step 5: Keep displaying temp → data until temp reaches to the last node
step 6: Finally display temp → data
step 7: Stop.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define  null 0
void push();
void pop();
void peep();
void display();
int ch,item,b;
struct node
{
int data;
struct node*next;
}*top=null,*top,*New,*temp;
void main()
{
printf("\n\t**DYNAMIC IMPLEMENTATION OF STACK**");
while(1)
  {
   printf("\nMenu:\n\t1.push\n\t2.pop\n\t3.peep\n\t4.display\n\t5.exit");
   printf("\n Enter you choice");
   scanf("%d",&ch);
   switch(ch)
     {
      case 1:
       push();
        break;
       case 2:
        pop();
      break;
      case 3:
```

```
        peep();
        break;
        case 4:
        display();
        break;
        case 5:
        exit(0);
        default:
         printf("try again");
        }
    }
}
void push()
{
  New=(struct node*)malloc(sizeof(struct node));
       if(New==null)
          printf("\nout of space");
          else
          {
              printf("\nEnter the element to be insert:");
             scanf("%d",&item);
             New->data=item;
             New->next=top;
             top=New;
          }
}
void pop()
{
   if (top==null)
   printf("stack is empty");
   else
   {
      temp=top;
      top=temp->next;
      printf("deleted element is:%d",temp->data);
      free(temp);
   }
}
void peep()
{
   int count=0;
   if (top==null)
   printf("stack is empty");
```

```
   else
   {
      printf("\nEnter the element to search:");
      scanf("%d",&item);
      temp=top;
      do
      {
         count=count+1;
         if(item==temp->data)
           {
            printf("element found at position");
            printf("%d",count);
            break;
           }
         else
          temp=temp->next;
       }
         while(temp!=null);
if(temp==null)
printf("element not found");
   }}
void display()
{
  if (top==null)
  printf("stack is empty");
  else
    {
      temp=top;
      printf("\nThe elements in the list are");
      while(temp!=null)
        {
          printf("\n%d",temp->data);
          temp=temp->next;
        }}}
```

OUTPUT:

```
      **DYNAMIC IMPLEMENTATION OF STACK**
Menu:
      1.push
      2.pop
      3.peep
      4.display
      5.exit
 Enter you choice1
```

```
Enter the element to be insert:1
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice1
Enter the element to be insert:2
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice1
Enter the element to be insert:3
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice1
Enter the element to be insert:4
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice4
The elements in the list are
4
3
2
1
Menu:
1.push
2.pop
3.peep
 4.display
```

```
        5.exit
 Enter you choice2
deleted element is:4
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice4
The elements in the list are
3
2
1
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice3
Enter the element to search:4
element not found
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice3
Enter the element to search:2
element found at position2
Menu:
        1.push
        2.pop
        3.peep
        4.display
        5.exit
 Enter you choice5
```

RESULT:

Thus the C program to implement stack using dynamic implementation was written and output is verified successfully.

EX.NO : 2C	LINKED LIST IMPLEMENTATION OF LIST ADT
DATE:	

AIM:

To write a C Program for dynamic implementation of singly linked list.

ALGORITHM:

Step 1 - Start

Step 2 - Include all the header files which are used in the program and define null as 0.

Step 3 - Declare all the functions used in linked list implementation.

Step 4 - In main method, display menu with list of operations and make suitable function calls to perform operation using switch case depends on the users choice.

Step 5 - Stop

CREATE()

Step 1 - Start

Step 2 – Define a New Node structure with data and next members.

Step 3 - Define a Node pointer 'head' and set it to null.

Step 4 – Stop

INSBEGIN ( )

Step 1 - Start
Step 2 - Create a newnode with given value.
Step 3 - Check whether list is Empty (head == null)
Step 4 - If it is Empty then, set head =tail= newnode.
Step 5 – else,Set 'newnode → next =head', 'head = newnode'
Step 7 - Stop

INSEND ( )
Step 1 -- start
Step 2 - create a newNode with given value.
Step 3 - check whether list is empty (head == null).
Step 4 - if it is empty then, set head =tail= newNode.
Step 5 - else, assign tail→next=newNode.

Step 6 – Assign tail=New.

Step 7 – Stop

INSPOS ( )

step 1 - Start

step 2 - Create a newNode with given value.

step 3 - Check whether list is Empty (head == null)

step 4 - If it is Empty then, set head =tail= newNode.

step 5 - else, define two Node pointers 'temp' and 'curr' initialize 'temp' with head.

step 6 - Keep moving the temp until it reaches to the exact position and every time set 'curr = temp' before moving the 'temp' to its next node.

step 7 - If temp is reached to the exact node after which we want to insert the newNode then assign 'curr→next=newNode' and 'newNode→next=temp'.

step 8 - Stop.

DELBEGIN ( )

step 1 - Start.

step 2 - Check whether list is Empty (head == null)

step 3 - If it is Empty then print list is empty.

step 4 - else, temp=head and set head=temp→next and free temp.

step 5 - Stop.

DELEND ( )

Step 1 - Start.

Step 2 - Check whether list is Empty (head == null)

Step 3 - If it is Empty then print list is empty.

Step 4 - else, temp=head and traverse the list until tail and free temp.

Step 5 - Stop.

DELPOS ( )

Step 1 - Start.

Step 2 - Check whether list is Empty (head == null)

Step 3 - If it is Empty then, display 'List is Empty!!! Deletion is not possible.

Step 4 - else, define two Node pointers 'temp' and 'curr' and initialize 'temp' with head.

Step 5 - Keep moving the temp until it reaches to the exact node to be deleted or to the last node. And every time set 'curr = temp' before moving the 'temp' to its next node.

Step 6 - If it is reached to the last node then display 'Given node not found in the list! Deletion not possible And terminate the function.

Step 7 - If it is reached to the exact node which we want to delete, then set curr→ next = temp → next and delete temp (free(temp)).

Step 8 - Stop.

FIND( )

Step 1 - Start

Step 2 - Set count=0

Step 3 –If (head==null),then display “LIST IS EMPTY”.

Step 4 – Else, read element x to search

Step 5 - traverse the list using while loop

Step 6 - compare the element x with list elements

Step 7 - If x==list elements print “element found” else print “element not found”
Step 8 - Stop.

DISPLAY ( )

Step 1 - Start.
Step 2 - check whether list is empty (head == null)
Step 3 - if it is empty, then display 'list is empty!!!' And terminate the function.
Step 4 - else, define a node pointer 'temp' and initialize with head.
Step 5 - keep displaying temp → data until temp reaches to the last node
Step 6 - finally display temp → data
Step 7 - Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define null 0
void create();
void InsBegin();
void InsEnd();
void InsPos();
void DelBegin();
void DelEnd();
void DelPos();
void display();
void find();
int x,i,ch,n,pos;
struct node
{
   int data;
   struct node *next;
}*head=null,*New,*temp,*curr,*tail=null;
void main()
{

   printf("\n\t*****SINGLY LINKED LIST*****");
   while(1)
   {

printf("\nMenu:\n\t1.create\n\t2.InsBegin\n\t3.InsEnd\n\t4.InsPos\n\t5.DelBegin\n\t6.DelEnd\n\t7.Del
Pos\n\t8.display\n\t9.find\n\t10.exit");
      printf("\nEnter your choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
```

```
            create();
            break;
        case 2:
            InsBegin();
            break;
        case 3:
            InsEnd();
            break;
        case 4:
            InsPos();
            break;
        case 5:
            DelBegin();
            break;
        case 6:
            DelEnd();
            break;
        case 7:
            DePos();
            break;
        case 8:
            display();
            break;
        case 9:
            find();
            break;
        case 10:
            exit(0);
        default:
            printf("Try Again");
            break;
        }
    }
}
void create()
{
    printf("Enter the number of nodes to be created:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        New=(struct node*)malloc(sizeof(struct node));
        if(New==null)
        printf("\nOut of space");
        else
```

```
        {
         printf("\nEnter the data:");
         scanf("%d",&x);
         New->data=x;
         New->next=null;
         if(head==null)
           head=New;
         else
         {
          temp=head;
          while(temp->next!=null)
          {
             temp=temp->next;
          }
          temp->next=New;
          New->next=null;
          tail=New;
         }
        }
    }
}
void InsBegin()
{
       New=(struct node*)malloc(sizeof(struct node));
       if(New==null)
          printf("\nOut of space");
       else
       {
         printf("Enter the element to be inserted:");
         scanf("%d",&x);
         New->data=x;
         New->next=head;
         head=New;
       }
}
void InsEnd()
{
   if(pos>=n)
      printf("Position is out of range");
   else
   {
      New=(struct node*)malloc(sizeof(struct node));
      if(New==null)
         printf("\nOut of space");
      else
```

```
        {
         printf("Enter the element to be inserted:");
         scanf("%d",&x);
         New->data=x;
         New->next=null;
         tail->next=New;
         New=tail;
        }
    }
}
void InsPos()
{
    int count=0;
    temp=head;
    while(temp!=null)
    {
       count=count+1;
       temp=temp->next;
    }
    printf("enter the position in which the element to be inserted:\n");
    scanf("%d",&pos);
    if(pos<=1||pos>count)
     printf("Position is out of range");
    else
    {
       New=(struct node*)malloc(sizeof(struct node));
       if(New==null)
          printf("\nOut of space");
       else
       {
         printf("Enter the element to be inserted:");
         scanf("%d",&x);
         New->data=x;
         New->next=null;
         temp=head;
         for(i=1;i<pos;i++)
         {
            curr=temp;
            temp=temp->next;
         }
         curr->next=New;
         New->next=temp;
       }
    }
}
```

```
void DelBegin()
{
   {
      if(head==null)
         printf("List is empty");
      else
      {
         temp=head;
         head=temp->next;
         printf(“Deleted Element:%d”,temp->data);
        free(temp);
      }
   }
}
void DelEnd()
{
      if(head==null)
         printf("List is empty");
      else
      {
         temp=head;
         while(temp->next!=null)
         {
            curr=temp;
            temp=temp->next;
         }
         printf(“Deleted Element:%d”,temp->data);
         free(temp);
         curr->next=null;
         tail=curr;
      }
}
void DelPos()
{
   int count=0;
   temp=head;
   while(temp!=null)
   {
      count=count+1;
      temp=temp->next;
   }
   printf("enter the position in which the element to be inserted:\n");
   scanf("%d",&pos);
   if(pos<=1||pos>count)
    printf("Position is out of range");
```

```
    else
    {
       if(head==null)
          printf("\nList is empty");
       else
       {
          printf("Enter the position of element to be deleted:");
          scanf("%d",&pos);
          temp=head;
          for(i=1;i<pos;i++)
          {
             curr=temp;
             temp=temp->next;
          }
          curr->next=temp->next;
          printf("Deleted Element:%d",temp->data);
          free(temp);
       }
    }
}
void display()
{
   if(head==null)
      printf("\nList is empty");
   else
   {
      temp=head;
      printf("the elements in the list are:");
      while(temp!=null)
      {
         printf("\t%d",temp->data);
         temp=temp->next;
      }
   }
}
void find()
{
   int count=0;
   if(head==null)
      printf("\nList is empty");
   else
   {
   printf("\nEnter the element:");
   scanf("\t%d",&x);
   temp=head;
```

```
            while(temp!=null)
            {
               count=count+1;
               if(x==temp->data)
               {
                 printf("Element found at the position:");
                 printf("%d",count);
                 break;
               }
              else
              {
                 temp=temp->next;
              }
              if(temp==null)
              {
                printf("Element not found");
              }
            }
            }
        }
```

OUTPUT:

```
                *****SINGLY LINKED LIST*****

        MENU:-
        1.create
        2.InsBegin
        3.InsEnd
        4.InsPos
        5.DelBegin
        6.DelEnd
        7.DelPos
        8.display
        9.find
        10.exit
        Enter your choice:1

        Enter the no.of.nodes to be created:4
        Enter the data:3
        Enter the data:5
        Enter the data:7
        Enter the data:9

        MENU:-
        1.create
```

2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are: 3 5 7 9

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:2

Enter the Element to be inserted:6

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:3

Enter the Element to be inserted:1

MENU:-
1.create
2.InsBegin

```
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are:    6      3      5      7      9      1


MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:4

Enter the position to be inserted:5
Enter the Element to be inserted:8

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are:    6      3      5      7      8      9      1

MENU:-
1.create
2.InsBegin
```

3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:5
Deleted Element:6

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:6
Deleted Element:1

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are: 3 5 7 8 9

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin

6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:7
Enter the position the element to be inserted:3
Deleted Element:7

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are: 3 5 8 9

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:9
Enter the element to be found:9
Element is found at the position:4

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin

6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice: 9
Enter the element to be found:7
Element not found

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:10

RESULT:

Thus the C program to implement singly linked list using dynamic implementation has been executed and output is verified successfully**.**

EX.NO : 2D	LINKED LIST IMPLEMENTATION OF LIST ADT
DATE:	

AIM:

To write a C Program for dynamic implementation of doubly linked list.

ALGORITHM:

Step 1 - Start

Step 2 - Include all the header files which are used in the program and define null as 0.

Step 3 - Declare all the functions used in linked list implementation.

Step 4 - In main method, display menu with list of operations and make suitable function calls to perform operation using switch case depends on the users choice.

Step 5 - Stop

CREATE()

Step 1 - Start

Step 2 – Define a New Node structure with data and next members.

Step 3 - Define a Node pointer 'head' and set it to null.

Step 4 – Stop

INSBEGIN ( )

Step 1 - Start
Step 2 - Create a newnode with given value.
Step 3 - Check whether list is Empty (head == null)
Step 4 - If it is Empty then, set head =tail= newnode.
Step 5 - Else, Set 'newnode → next =head', 'newnode → prev =null' and 'head = newnode'
Step 6 - Stop

INSEND ( )
Step 1 -- start
Step 2 - create a newNode with given value.
Step 3 - check whether list is empty (head == null).
Step 4 - if it is empty then, set head =tail= newNode.
Step 5 - else, Set 'newnode → prev =tail','newnode → next =null' and 'tail→next=newNode'.

Step 6 – Assign tail=New.

Step 7 – Stop

INSPOS ( )

step 1 - Start

step 2 - Create a newNode with given value.

step 3 - Check whether list is Empty (head == null)

step 4 - If it is Empty then, set head =tail= newNode.

step 5 - else, define two Node pointers 'temp' and 'curr' initialize 'temp' with head.

step 6 - Keep moving the temp until it reaches to the exact position and every time set 'curr = temp' before moving the 'temp' to its next node.

step 7 - If temp is reached to the exact node after which we want to insert the newNode then, set 'curr→next=newNode', 'newnode → prev =curr', ' newNode→next=temp' and 'temp→prev=newNode'.

step 8 - Stop.

DELBEGIN ( )

step 1 - Start.

step 2 - Check whether list is Empty (head == null)

step 3 - If it is Empty then print list is empty.

step 4 - else, temp=head and set head=temp→next and free temp.

step 5 - Stop.

DELEND ( )

Step 1 - Start.

Step 2 - Check whether list is Empty (head == null)

Step 3 - If it is Empty then print list is empty.

Step 4 - else, temp=head and traverse the list until tail and free temp.

Step 5 - Stop.

DELPOS ( )

Step 1 - Start.

Step 2 - Check whether list is Empty (head == null)

Step 3 - If it is Empty then, display 'List is Empty!!! Deletion is not possible.

Step 4 - else, define three Node pointers 'N', 'temp' and 'curr' and initialize 'temp' with head.

Step 5 - Keep moving the temp until it reaches to the exact node to be deleted or to the last node. And every time set 'curr = temp' before moving the 'temp' to its next node.

Step 6 - If it is reached to the last node then display 'Given node not found in the list! Deletion not possible And terminate the function.

Step 7 - If it is reached to the exact node which we want to delete, then set 'curr→ next = temp → next', 'N=temp→next', 'N→prev=curr' and delete temp (free(temp)).

Step 8 - Stop.

FIND ( )

Step 1 - Start

Step 2 - Set count=0

Step 3 –If (head==null),then display “LIST IS EMPTY”.

Step 4 – Else, read element x to search

Step 5 - traverse the list using while loop

Step 6 - compare the element x with list elements
Sep 7 – If x==list element print “element found” else print “element not found”
Step 8 - Stop.

DISPLAY ( )

Step 1 - Start.
Step 2 - check whether list is empty (head == null)
Step 3 - if it is empty, then display 'list is empty!!!' And terminate the function.
Step 4 - else, define a node pointer 'temp' and initialize with head.
Step 5 - keep displaying temp → data until temp reaches to the last node
Step 6 - finally display temp → data
Step 7 - Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define null 0
void create();
void InsertatBegin();
void InsertatEnd();
void InsertatPos();
void DelatBegin();
void DelatEnd();
void DelatPos();
void display();
void find();
int x,i,ch,n,pos;
struct node
{
   int data;
   struct node *next, *prev;
}*head=null,*N,*New,*temp,*curr,*tail=null;
void main()
{
   printf("\n\t****DOUBLY LINKED LIST****");
   while(1)
   {

printf("\nMenu:\n\t1.create\n\t2.InsertatBegin\n\t3.InsertatEnd\n\t4.InsertatPos\n\t5.DelatBegin\n\t6.D
elatEnd\n\t7.DelatPos\n\t8.display\n\t9.find\n\t10.exit");
      printf("\nEnter your choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         create();
```

```
            break;
        case 2:
            InsertatBegin();
            break;
        case 3:
            InsertatEnd();
            break;
        case 4:
            InsertatPos();
            break;
        case 5:
            DelatBegin();
            break;
        case 6:
            DelatEnd();
            break;
        case 7:
            DelatPos();
            break;
        case 8:
            display();
            break;
        case 9:
            find();
            break;
        case 10:
            exit(0);
        default:
            printf("Try Again");
            break;
        }
    }
}
void create()
{
    printf("Enter the number of nodes to be created:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        New=(struct node*)malloc(sizeof(struct node));
        if(New==null)
        printf("\nOut of space");
        else
        {
         printf("\nEnter the data:");
```

```
        scanf("%d",&x);
        New->data=x;
        New->prev=null;
        New->next=null;
        if(head==null)
        {
          head=New;
          tail=New;
        }
        else
        {
         temp=head;
         while(temp->next!=null)
         {
            temp=temp->next;
         }
         temp->next=New;
         New->prev=temp;
         New->next=null;
         tail=New;
        }
       }
    }
}
void InsertatBegin()
{
       New=(struct node*)malloc(sizeof(struct node));
       if(New==null)
          printf("\nOut of space");
       else
       {
        printf("Enter the element to be inserted:");
        scanf("%d",&x);
        New->data=x;
        New->prev=null;
        New->next=head;
        head->prev=New;
        head=New;
       }

}
void InsertatEnd()
{
       New=(struct node*)malloc(sizeof(struct node));
       if(New==null)
```

```
            printf("\nOut of space");
        else
        {
          printf("Enter the element to be inserted:");
          scanf("%d",&x);
          New->data=x;
          New->next=null;
          tail->next=New;
          New->prev=tail;
          tail=New;
        }
}
void InsertatPos()
{
   int count=0;
   temp=head;
   while(temp!=null)
   {
      count=count+1;
      temp=temp->next;
   }
   printf("enter the position in which the element to be inserted:\n");
   scanf("%d",&pos);
   if(pos<=1||pos>count)
    printf("Position is out of range");
   else
   {
      New=(struct node*)malloc(sizeof(struct node));
      if(New==null)
         printf("\nOut of space");
      else
      {
        printf("Enter the element to be inserted:");
        scanf("%d",&x);
        New->data=x;
        New->prev=null;
        New->next=null;
        temp=head;
        for(i=1;i<pos;i++)
        {
           curr=temp;
           temp=temp->next;
        }
        curr->next=New;
        New->prev=curr;
```

```
            New->next=temp;
            temp->prev=New;
        }
    }
}
void DelatBegin()
{
    if(head==null)
        printf("List is empty");
    else
    {
        temp=head;
        head=temp->next;
        head->prev=null;
        printf("Deleted element:%d",temp->data);
        free(temp);
    }
}
void DelatEnd()
{
    if(head==null)
        printf("List is empty");
    else
    {
        temp=head;
        while(temp->next!=null)
        {
            curr=temp;
            temp=temp->next;
        }
        printf("\nDeleted element:%d",temp->data);
        free(temp);
        curr->next=null;
        tail=curr;
    }
}
void DelatPos()
{
  int count=0;
  temp=head;
  while(temp!=null)
  {
    count=count+1;
    temp=temp->next;
  }
```

```
    if(pos<=1||pos>count)
     printf("Position is out of range");
    else
    {
       if(head==null)
          printf("\nList is empty");
       else
       {
          printf("Enter the position of element to be deleted:");
          scanf("%d",&pos);
          temp=head;
          for(i=1;i<pos;i++)
          {
             curr=temp;
             temp=temp->next;
          }
          curr->next=temp->next;
          N=temp->next;
          N->prev=curr;
          printf("\nDeleted element:%d",temp->data);
          free(temp);
       }
    }
}
void display()
{
    if(head==null)
       printf("\nList is empty");
    else
    {
       temp=head;
       printf("the elements in the list are:");
       while(temp!=null)
       {
          printf("\t%d",temp->data);
          temp=temp->next;
       }
    }
}
void find()
{
    int count=0;
    if(head==null)
       printf("\nList is empty");
    else
```

```
            {
            printf("\nEnter the element:");
            scanf("\t%d",&x);
            temp=head;
            while(temp!=null)
             {
               count=count+1;
               if(x==temp->data)
               {
                 printf("Element found at the position:");
                 printf("%d",count);
                 break;
               }
              else
               {
                  temp=temp->next;
               }
              if(temp==null)
               {
                printf("Element not found");
               }
             }
            }
}
```

OUTPUT:

```
                *****DOUBLY LINKED LIST*****

          MENU:-
          1.create
          2.InsBegin
          3.InsEnd
          4.InsPos
          5.DelBegin
          6.DelEnd
          7.DelPos
          8.display
          9.find
          10.exit
          Enter your choice:1

          Enter the no.of.nodes to be created:3
          Enter the data:2
          Enter the data:5
```

```
Enter the data:8

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are:    2        5        8

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:2

Enter the Element to be inserted:7

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:3

Enter the Element to be inserted:6
```

```
MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are:    7      2      3      8      6


MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:4

Enter the position to be inserted:4
Enter the Element to be inserted:3

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are:    7      2      5      3      8     6
```

```
MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:5
Deleted Element:7

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:6
Deleted Element:6

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are:    2       5       3       8

MENU:-
1.create
```

2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:7
Enter the position the element to be inserted:2
Deleted Element:5

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:8
The elements in the list are: 2 3 8

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:9
Enter the element to be found:3
Element is found at the position:2

MENU:-
1.create
2.InsBegin

3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice: 9
Enter the element to be found:5
Element not found

MENU:-
1.create
2.InsBegin
3.InsEnd
4.InsPos
5.DelBegin
6.DelEnd
7.DelPos
8.display
9.find
10.exit
Enter your choice:10

RESULT:

Thus the C program to implement doubly linked list using dynamic implementation has been executed and output is verified successfully**.**

EX.NO. : 2E	IMPLEMENTATION OF SINGLY
DATE:	CIRCULAR LINKED ADT

AIM:

To write a c program for dynamic implementation of list (circular singly linked list).

ALGORITHM:

step1: Start.

step2: Using while loop perform varies operations in linked list repeatedly until return (exit) statement is executed.
step3: Read choice.

step4: Switch case is used to select what operations to be performs and perform the operations according to the case.
step5: Stop.

CREATE ( )

step 1: Start

step 2: Define a Node structure with two members data and next
step 3: Define a Node pointer 'head' and set it to null.
step 4: Stop

INSBEGIN ( )

step 1:Start

step 2: Create a newNode with given value.

step 3: Check whether list is Empty (head == null)
step 4: If it is Empty then, set head =tail= newNode.
step 5: If it is Not Empty then,
step 6: Set 'newNode → next =head', 'head = newNode'
step 7: Stop

INSEND ( )

step 1:Start

step 2: Create a newNode with given value.

step 3: Check whether list is Empty (head == null).

step 4: If it is Empty then, set set head =tail= newNode.

step 5: If it is Not Empty then, define a node pointer temp and initialize with head.

step 6: Keep moving the temp to its next node until it reaches to the last node in the list (until temp →

.

next == head).
step 7: Set temp → next = newNode and newNode → next = head.
step 8:Stop.

INSPOS ( )

step 1:Start

step 2: Create a newNode with given value.

step 3: Check whether list is Empty (head == null)
step 4: If it is Empty then, set head =tail= newNode.
step 5: If it is Not Empty then, define a node pointer temp and initialize with head.

step 6: Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).
step 7: Every time check whether temp is reached to the last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.
step 8: If temp is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).
step 9: If temp is last node then set temp → next = newNode and newNode → next = head.

step 10: If temp is not last node then set newNode → next = temp → next and temp → next = newNode.
step 11: Stop.

DELBEGIN ( )

step 1: Start.

step 2: Check whether list is Empty (head

== null)step 3: If it is Empty then print list is
empty.
step 4: else,head=temp and set temp→next=head and free temp.
step 5: Stop.

DELEND ( )

step 1: Start.

step 2: Check whether list is Empty (head == null)
step 3: If it is Empty then print list is empty.
step 4: else,temp=head and traverse the list until tail,now free tail.

step 5: Stop.

DELPOS ( )

step 1: Start.

step 2: Check whether list is Empty (head == null)

step 3: If it is Empty then, display 'List is Empty!!! Deletion is not possible' and Data terminate the

function.
step 4: If it is Not Empty then, define two Node pointers 'temp' and 'curr' and initialize 'temp' with head.
step 5: Keep moving the temp until it reaches to the exact node to be deleted or to the last node. And every time set 'curr = temp' before moving the 'temp' to its next node.
step 6: If it is reached to the last node then display 'Given node not found in the list! Deletion not possible And terminate the function.
step 7: If it is reached to the exact node which we want to delete, then check whether list is having only one node (temp → next== head)
step 8: If list has only one node and that is the node to be deleted then set head = null and delete temp (free(temp)).
step 9: If list contains multiple nodes then check whether temp is the first node in the list (temp == head).
step 10: If temp is the first node then set curr = head and keep moving curr to its next node until curr reaches to the last node. Then set head = head →next, curr→ next = head and delete temp1.
step 11: If temp is not first node then check whether it is last node in the list
(temp → next == head).
step 12: If temp is last node then set curr→ next = head and delete temp (free(temp1)).

step 13: If temp is not first node and not last node then set curr→ next = temp → next and delete temp (free(temp)).
step 14: Stop.

SEARCH ( )

Step 1:Start

Step 2:Set count=0

Step 3:Read element x to search

Step 4: traverse the list using while loop

Step 5:compare the element x with list elements

Step 6:If x==list elements print “element found” else print“element not found”

Step 7:Stop.

DISPLAY ( )

step 1: start.

step 2: Check whether list is Empty (head == null)

step 3: If it is Empty, then display 'List is Empty!!!' and terminate the function.

step 4: If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
step 5: Keep displaying temp → data until temp reaches to the last node
step 6: Finally display temp → data
step 7: Stop.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define  null 0
void create();
void insbegin();
void insend();
void insany();
void delbegin();
void delend();
void delany();
void search();
void display();
int  n,ch,item,i,pos;
struct node
{
int data;
struct node*next;
}*head=null,*New,*temp,*curr,*tail=null;
void main()
{
 printf("\n\t*SINGLY CIRCULAR LINKED LIST*");
 while(1)
   {
    printf("\nMenu:\n\t1.create list\n\t2.insert at begin\n\t3.insert at end\n\t4.insert at position\n\t5.del at
begin\n\t6.del at end\n\t7.del at position\n\t8.search\n\t9.display\n\t10.exit");
    printf("\n Enter you choice");
    scanf("%d",&ch);
    switch(ch)
       {
        case 1:
          create();
        break;
        case 2:
          insbegin();
          break;
        case 3:
        insend();
        break;
        case 4:
        insany();
        break;
        case 5:
          delbegin();
```

```
        break;
      case 6:
      delend();
      break;
      case 7:
      delany();
      break;
      case 8:
      search();
      break;
      case 9:
      display();
      break;
      case 10:
      exit(0);
      default:
       printf("try again");
      }
  }
}
void create()
 {
   printf("Enter the size of the list:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      New=(struct node*)malloc(sizeof(struct node));
      if(New==null)
         printf("\nout of space");
      else
      {
         printf("\nEnter the data:");
         scanf("%d",&item);
         New->data=item;
         New->next=null;
         if(head==null)
         {
          head=New;
          tail=New;
          tail->next=head;
         }
         else
         {
          tail->next=New;
```

```
            New->next=head;
            tail=New;
           }
        }
    }
}
void insbegin()
{
    New=(struct node*)malloc(sizeof(struct node));
    if(New==null)
        printf("\nout of space");
    else
    {
      printf("\nEnter the elments to be insert:");
      scanf("%d",&item);
      New->data=item;
      New->next=head;
      head=New;
      tail->next=head;
    }
}
void insend()
{
    New=(struct node*)malloc(sizeof(struct node));
    if(New==null)
      printf("\nout of space");
    else
    {
     printf("\nEnter the elments to be insert:");
     scanf("%d",&item);
     New->data=item;
     tail->next=New;
     New->next=head;
     tail=New;
    }
}
void insany()
{
    int count=1;
    temp=head->next;
    while(temp!=head)
    {
     count=count+1;
```

```
    temp=temp->next;
   }
    printf("\nEnter the position:");
    scanf("%d",&pos);
    if(pos<1||pos>count+1)
    {
     printf("position out of range");
    }
    else
          {
           if(pos==1)
              insbegin();
           else if(pos==count+1)
              insend();
           else
              {
            New=(struct node*)malloc(sizeof(struct node));
            if(New==null)
             printf("\nout of space");
            else
               {
             printf("\nEnter the elments to be insert:");
             scanf("%d",&item);
             New->data=item;
             New->next=null;
             temp=head;
             for(i=1;i<pos;i++)
                  {
                 curr=temp;
                 temp=temp->next;
                   }
             curr->next=New;
             New->next=temp;
                 }
              }
          }
}
void delbegin()
{
   if(head==null)
      printf("\nlist is empty");
   else
   {
```

```
        temp=head;
        printf("deleted element is:%d",temp->data);
        head=temp->next;
        free(temp);
        tail->next=head;
    }
}
void delend()
{
    if(head==null)
        printf("\nlist is empty");
    else
    {
        temp=head;
        while(temp->next!=head)
        {
            curr=temp;
            temp=temp->next;
        }
        printf("deleted element is:%d",temp->data);
        curr->next=head;
        tail=curr;
        free(temp);
        tail->next=head;
    }
}
void delany()
{
     int count=1;
                temp=head;
                while(temp!=tail)
                {
                    count=count+1;
                    temp=temp->next;
                }
                 printf("\nEnter the position:");
                scanf("%d",&pos);
                if(pos<1||pos>count)
                {
                    printf("position out of range");
                }
                else
```

```
            {

   if(head==null)
      printf("\nlist is empty");
   else
   {
      if(pos==1)
       delbegin();
       else if(pos==count)
       delend();
       else
       {
            temp=head;
            for(i=1;i<pos;i++)
            {
               curr=temp;
               temp=temp->next;
            }
            curr->next=temp->next;
            printf("deleted element is:%d",temp->data);
            free(temp);
       }
   }

   }
}
void search()
{
   int count=1;
   if(head==null)
      printf("\nlist is empty");
   else
   {
      printf("\nEnter the elements to search:");
      scanf("%d",&item);

      if(item==head->data)
      printf("element found at position1");
      else
      {
      temp=head->next;
      do
       {
```

```
        count=count+1;
        if(item==temp->data)
         {
      printf("element found at position");
      printf("%d",count);
      break;
           }
            else
            temp=temp->next;
       }
      while(temp!=head);
       if(temp==head)
       printf("element not found");
      }
   }
}
void display()
{
if(head==null)
printf("\n The list is empty");
else
  {
   temp=head;
   printf("\nThe elements in the list are");
    while(temp!=tail)
     {
      printf("\n%d",temp->data);
      temp=temp->next;
     }
 printf("\n%d" ,temp->data);
  }
}
```

OUTPUT:

SINGLY CIRCULAR LINKED LIST

Menu:

1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search

9.display
10.exit
Enter you choice1
Enter the size of the list:4
Enter the data:11
Enter the data:22
Enter the data:33
Enter the data:44
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice2
Enter the elments to be insert:10
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice3
Enter the elments to be insert:55
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit

```
 Enter you choice4
Enter the position:4
Enter the elments to be insert:25

Menu:
        1.create list
        2.insert at begin
        3.insert at end
        4.insert at position
        5.del at begin
        6.del at end
        7.del at position
        8.search
        9.display
        10.exit
 Enter you choice9
The elements in the list are
10
11
22
25
33
44
55
Menu:
        1.create list
        2.insert at begin
        3.insert at end
        4.insert at position
        5.del at begin
        6.del at end
        7.del at position
        8.search
        9.display
        10.exit
 Enter you choice5
deleted element is:10
Menu:
        1.create list
        2.insert at begin
        3.insert at end
        4.insert at position
        5.del at begin
        6.del at end
        7.del at position
```

8.search
9.display
10.exit
Enter you choice6
deleted element is:55
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice7

Enter the position:3
deleted element is:25
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice9

The elements in the list are
11
22
33
44
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end

7.del at position
8.search
9.display
10.exit
Enter you choice8

Enter the elements to search:25
element not found
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice8

Enter the elements to search:33
element found at position3
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice10

RESULT:

Thus the C program to implement singly circular linked list using dynamic implementation was written and output is verified successfully.

.

EX.NO:2F	IMPLEMENTATION OF DOUBLY
DATE:	LINKED LIST

AIM:

To write a c program for dynamic implementation of list (circular doubly linked list).

ALGORITHM:

step1: Start.

step2: Using while loop perform varies operations in linked list repeatedly until return (exit) statement is executed.

step3: Read choice.

step4: Switch case is used to select what operations to be performs and perform the operations according to the case.

step5: Stop.

CREATE ( )

step 1: Start

step 2: Define a Node structure with three members data, next and prev

step 3: Define a Node pointer 'head' and set it to null.

step 4: Stop

INSBEGIN ( )

step 1:Start

step 2: Create a newNode with given value.

step 3: Check whether list is Empty (head == null)

step 4: If it is Empty then, set head =tail= newNode.

step 5: If it is Not Empty then,

step 6: Set 'newNode → next =head', newNode→prev=tail, 'head = newNode'

step 7: Stop

INSEND ( )

step 1:Start

step 2: Create a newNode with given value.

step 3: Check whether list is Empty (head == null).

step 4: If it is Empty then, set head =tail= newNode.

step 5: If it is Not Empty then, define a node pointer temp and initialize with head.

step 6: Keep moving the temp to its next node until it reaches to the last node in the list (until temp →

next == head).
step 7: Set temp → next = newNode and newNode → next = head and newNode→prev=temp.
step 8:Stop.

INSPOS ( )

step 1:Start

step 2: Create a newNode with given value.

step 3: Check whether list is Empty (head == null)
step 4: If it is Empty then, set head =tail= newNode.
step 5: If it is Not Empty then, define a node pointer temp and initialize with head.

step 6: Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).
step 7: Every time check whether temp is reached to the last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.
step 8: If temp is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).
step 9: If temp is last node then set temp → next = newNode and newNode → next = head and newNode→prev=temp.
step 10: If temp is not last node then set newNode → next = temp → next and temp → next = newNode.
step 11: Stop.

DELBEGIN ( )

step 1: Start.

step 2: Check whether list is Empty (head == null)
step 3: If it is Empty then print list is empty.
step 4: else,head=temp and set temp→next=head and free temp.
step 5: Stop.

DELEND ( )

step 1: Start.

step 2: Check whether list is Empty (head == null)
step 3: If it is Empty then print list is empty.

step 4: else,temp=head and traverse the list until tail,now free tail.
step 5: Stop.

DELPOS ( )

step 1: Start.

step 2: Check whether list is Empty (head == null)

step 3: If it is Empty then, display 'List is Empty!!! Deletion is not possible' and Data terminate the function.
step 4: If it is Not Empty then, define two Node pointers 'temp' and 'curr' and initialize 'temp' with

head.
step 5: Keep moving the temp until it reaches to the exact node to be deleted or to the last node. And every time set 'curr = temp' before moving the 'temp' to its next node.
step 6: If it is reached to the last node then display 'Given node not found in the list! Deletion not possible And terminate the function.
step 7: If it is reached to the exact node which we want to delete, then check whether list is having only one node (temp → next== head)
step 8: If list has only one node and that is the node to be deleted then set head = null and delete temp (free(temp)).
step 9: If list contains multiple nodes then check whether temp is the first node in the list (temp == head).
step 10: If temp is the first node then set curr = head and keep moving curr to its next
node until currreaches to the last node. Then set head = head →next, curr→ next = head and delete temp1.
step 11: If temp is not first node then check whether it is last node in the list
(temp → next == head).
step 12: If temp is last node then set curr→ next = head and delete temp (free(temp1)).

step 13: If temp is not first node and not last node then set curr→ next = temp → next and delete temp (free(temp)).
step 14: Stop.

SEARCH ( )

Step 1:Start

Step 2:Set count=0

Step 3:Read element x to search

Step 4: traverse the list using while loop

Step 5:compare the element x with list elements

Step 6:If x==list elements print “element found” else print“element not found”
Step 7:Stop.
DISPLAY ( )

step 1: start.

step 2: Check whether list is Empty (head == null)

step 3: If it is Empty, then display 'List is Empty!!!' and terminate the function.

step 4: If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
step 5: Keep displaying temp → data until temp reaches to the last node
step 6: Finally display temp → data
step 7: Stop.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define  null 0
```

```
void create();
void insbegin();
void insend();
void insany();
void delbegin();
void delend();
void delany();
void search();
void display();
int  n,ch,item,i,pos;
struct node
{
int data;
struct node *next;
struct node *prev;
}*head=null,*New,*temp,*curr,*tail=null;
void main()
{
  printf("\n\t*DOUBLY CIRCULAR LINKED LIST*");
  while(1)
    {
      printf("\nMenu:\n\t1.create list\n\t2.insert at begin\n\t3.insert at end\n\t4.insert at position\n\t5.del at
begin\n\t6.del at end\n\t7.del at position\n\t8.search\n\t9.display\n\t10.exit");
      printf("\n Enter you choice");
      scanf("%d",&ch);
      switch(ch)
         {
        case 1:
           create();
           break;
        case 2:
           insbegin();
           break;
        case 3:
           insend();
           break;
        case 4:
           insany();
           break;
        case 5:
           delbegin();
           break;
        case 6:
           delend();
```

```
            break;
        case 7:
            delany();
            break;
        case 8:
            search();
            break;
        case 9:
            display();
            break;
        case 10:
            exit(0);
        default:
             printf("try again");
            }
    }
}
void create()
{
    printf("Enter the size of the list:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)

    {
        New=(struct node*)malloc(sizeof(struct node));
        if(New==null)
            printf("\nout of space");
            else
            {
                printf("\nEnter the data:");
                scanf("%d",&item);
                New->data=item;
                New->next=null;
                New->prev=null;
                if(head==null)
                {
                    head=New;
                    tail=New;
                    tail->next=head;
                    head->prev=tail;
                }
                    else
                    {
                        tail->next=New;
```

```
                New->next=head;
                tail=New;
                head->prev=tail;
            }
        }
    }
}
void insbegin()
{
     New=(struct node*)malloc(sizeof(struct node));
      if(New==null)
         printf("\nout of space");
         else
         {
            printf("\nEnter the elments to be insert:");
            scanf("%d",&item);
            New->data=item;
            New->next=head;
            New->prev=tail;
            head=New;
            tail->next=head;
         }
}
void insend()
{
   New=(struct node*)malloc(sizeof(struct node));
      if(New==null)
         printf("\nout of space");
         else
         {
             printf("\nEnter the elments to be insert:");
            scanf("%d",&item);
            New->data=item;
            tail->next=New;
            New->next=head;
            New->prev=tail;
            tail=New;
         }
}
void insany()
{
   int count=1;
```

```
        temp=head->next;
        while(temp!=head)
            {
            count=count+1;
            temp=temp->next;
            }
         printf("\nEnter the position:");
        scanf("%d",&pos);
        if(pos<1||pos>count+1)
            {
            printf("position out of range");
            }
        else
        {
             if(pos==1)
           insbegin();
           else if(pos==count+1)
           insend();
        else
         {
New=(struct node*)malloc(sizeof(struct node));
   if(New==null)
      printf("\nout of space");
      else
      {
         printf("\nEnter the elments to be insert:");
         scanf("%d",&item);
         New->data=item;
         New->next=null;
         New->prev=null;
         temp=head;
         for(i=1;i<pos;i++)
         {
            curr=temp;
            temp=temp->next;
         }
         curr->next=New;
         New->next=temp;
         New->prev=curr;
           }
         }
      }
```

```
}
void delbegin()
{
   if(head==null)
      printf("\nlist is empty");
   else
   {
      temp=head;
       printf("deleted element is:%d",temp->data);
      head=temp->next;
      head->prev=tail;
      free(temp);
      tail->next=head;
   }
}
void delend()
{
   if(head==null)
      printf("\nlist is empty");
   else
   {
      temp=head;
      while(temp->next!=head)
      {
         curr=temp;
         temp=temp->next;
      }
      printf("deleted element is:%d",temp->data);
      curr->next=head;
      tail=curr;
      free(temp);
      tail->next=head;
   }
}
void delany()
{
       int count=1;
            temp=head;
            while(temp!=tail)
            {
               count=count+1;
               temp=temp->next;
```

```
            }
             printf("\nEnter the position:");
            scanf("%d",&pos);
            if(pos<1||pos>count)
               {
                printf("position out of range");
                }
             else
              {
              if(head==null)
              printf("\nlist is empty");
              else
             {
              if(pos==1)
                 delbegin();
              else if(pos==count)
                  delend();
              else
                {
                 temp=head;
                 for(i=1;i<pos;i++)
                  {
                   curr=temp;
                   temp=temp->next;
                  }
                 curr->next=temp->next;
                 printf("deleted element is:%d",temp->data);
                 free(temp);
                  }
                }
                 }
}
void search()
{
   int count=1;
   if(head==null)
      printf("\nlist is empty");
   else
   {
      printf("\nEnter the elements to search:");
      scanf("%d",&item);
      if(item==head->data)
      printf("element found at position1");
```

```
        else
        {
        temp=head->next;
        do
          {
           count=count+1;
           if(item==temp->data)
             {
                printf("element found at position");
                printf("%d",count);
                break;
             }
           else
              temp=temp->next;
           }
        while(temp!=head);
   if(temp==head)
   printf("element not found");
          }
      }
}
void display()
{
if(head==null)
printf("\n The list is empty");
else
  {
   temp=head;
   printf("\nThe elements in the list are");
    while(temp!=tail)
      {
       printf("\n%d",temp->data);
       temp=temp->next;
      }
 printf("\n%d" ,temp->data);
  }
}
```

OUTPUT:

```
*DOUBLY CIRCULAR LINKED LIST*
Menu:
        1.create list
        2.insert at begin
        3.insert at end
```

4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice1
Enter the size of the list:4
Enter the data:11
Enter the data:22
Enter the data:33
Enter the data:44
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice2
Enter the elments to be insert:10
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice3
Enter the elments to be insert:55
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin

```
        6.del at end
        7.del at position
        8.search
        9.display
        10.exit
 Enter you choice4
Enter the position:4
Enter the elments to be insert:56
Menu:
        1.create list
        2.insert at begin
        3.insert at end
        4.insert at position
        5.del at begin
        6.del at end
        7.del at position
        8.search
        9.display
        10.exit
 Enter you choice9
The elements in the list are
10
11
22
56
33
44
55
Menu:
        1.create list
        2.insert at begin
        3.insert at end
        4.insert at position
        5.del at begin
        6.del at end
        7.del at position
        8.search
        9.display
        10.exit
 Enter you choice5
deleted element is:10
Menu:
        1.create list
        2.insert at begin
```

3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice6
deleted element is:55
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice7
Enter the position:3
deleted element is:56
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice9
The elements in the list are
11
22
33
44
Menu:
1.create list

2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice8
Enter the elements to search:45
element not found
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice8
Enter the elements to search:44
element found at position4
Menu:
1.create list
2.insert at begin
3.insert at end
4.insert at position
5.del at begin
6.del at end
7.del at position
8.search
9.display
10.exit
Enter you choice10

RESULT:

Thus the C program to implement doubly circular linked list using dynamic implementation was written and output is verified successfully.

EX.NO :3	POLYNOMIAL MANIPULATION
DATE :	

AIM:

To write a C Program to perform polynomial manipulations.

ALGORITHM:

Step 1 - Start

Step 2 - Read the no% of terms in first list and second list .

Step 3 - Read the coefficient and powers of the terms using for loop.

Step 4 - using while loop, perform various operation in polynomial expression repeatedly until return statement is executed .

Step 5 - Read choice and use switch case to select what operation to be performed.

Step 7 - Stop

Insert ( )

Step 1- Start

Step 2 - Create a new node and assign the values that passed to the function .

Step 3 - New → coef= coeff , New→pow=power , New→next=null

Step 4 - IF (*head==null) set *head=New . ELSE

temp=*head;

WHILE(temp→next!=null)

temp=temp→next;

temp→next=New

Step 5 - Stop.

Addpoly()

Step 1- Start

Step 2- Assign temporary pointers to the first two list .

Step 3- Using WHILE loop with condition (temp1!=null && temp2 !=null)

(i) IF (temp1→pow > temp2>pow)

Insert(&head3,temp1→coeff,temp1→pow)

temp1=temp1→next;

(ii)ELSEIF (temp1→pow < temp2→pow)

Insert(&head3,temp2→coeff,temp2→pow)

temp 2=temp2→next.

(iii)ELSE

Insert(&head3,temp1→coeff+temp2→coeff,temp1→pow)

temp1=temp1→next

temp 2=temp2→next

Step 4- Using WHILE loop with condition (temp2!=null)

Insert(&head3,temp2→coeff,temp2→pow)

Temp2=temp2→next

Step 5- Stop.

Subpoly()

Step 1- Start

Step 2- Assign temporary pointers to the first two list .

Step 3- Using WHILE loop with condition (temp1!=null && temp2 !=null)

(i) IF (temp1→pow > temp2>pow)

Insert(&head3,temp1→coeff,temp1→pow)
Temp1=temp1→next;

(ii)ELSEIF (temp1→pow < temp2→pow)

Insert(&head3,temp2→coeff,temp2→pow)

temp 2=temp2→next.

(iii)ELSE

Insert(&head3,temp1→coeff-temp2→coeff,temp1→pow)

temp1=temp1→next

temp 2=temp2→next

Step 4- Using WHILE loop with condition (temp2!=null)

Insert(&head3,temp2→coeff,temp2→pow)

temp2=temp2→next

Step 5- Stop.

Multiplypoly()

Step 1- Start

Step 2- Assign two temporary pointer for heads.

Step 3- Using WHILE loop with condition (temp1!=null)

temp2=head2

Using the inner WHILE loop with condition (temp2!=null)

Insert(&head3,temp1→coef*temp2→coef,temp1→pow+temp2→pow);

temp2=temp2→next;

temp1=temp1→next;

Step 4- Assign one temporary pointer for head3

Step 5 - Using while loop with condition (temp→=next!=null)

(i)IF (temp→pow=temp→next→pow)

temp→coeff=temp→coef + temp→next→coef

N=temp→next

temp→next=temp→next→next

free(n);

(ii)ELSE

temp=temp→next

Step 7- Stop.

Display()

Step 1- Start

Step 2- Assign temp to head3 ;

Step 3- Print "The result is"

Step 4- Using while loop with condition

(i) IF(temp !=head3)

Print "+"

(ii) Print("%d x^ %d",temp→coef,temp→pow)

temp=temp→next

Step 5- Stop

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define null 0

struct node
{
   int coef,pow;
   struct node *next;
}*m,*curr,*n,*head3=null,*head1=null,*head2=null,*temp,*temp1,*temp2,*New;
```

```
void insertpoly(struct node**,int,int);
void addpoly(struct node *,struct node *h2);
void subpoly(struct node *,struct node *h2);
void multiplypoly(struct node *,struct node *);
void display(struct node *);

int a,b,no,coefficient,powers,ch;

int main()
{
   printf("\n\t\tPOLYNOMIAL MANIPULATION\n");
   printf("\n\n\t\tFirst expression");
   printf("\nEnter the number of terms in the first list:");
   scanf("%d",&a);
   for(int i=1;i<=a;i++)
  {
        printf("\nEnter the coefficient of %d term:",i);
        scanf("%d",&coefficient);
        printf("\nEnter the power of the %d term:",i);
        scanf("%d",&powers);
        insertpoly(&head1,coefficient,powers);
    }
    printf("\n\t\tSecond expression");
    printf("\nEnter the number of terms in the second list:");
    scanf("%d",&b);
   for(int i=1;i<=b;i++)
  {
          printf("\nEnter the coefficient of %d term:",i);
          scanf("%d",&coefficient);
          printf("\nEnter the power of the %d term:",i);
          scanf("%d",&powers);
          insertpoly(&head2,coefficient,powers);
   }
   while(1)
   {
     printf("\n\n\t\tMENU:-");
     printf("\n\n\t1.addition\n\t2.subtraction\n\t3.multiplication\n\t4.display\n\t5.exit");
     printf("\nEnter your choice:");
     scanf("%d",&ch);
     switch(ch)
     {

     case 1:
       addpoly(head1,head2);
```

```
        break;
    case 2:
        subpoly(head1,head2);
        break;
    case 3:
        multiplypoly(head1,head2);
        break;
    case 4:
        printf("\n Enter the list you want too display (1/2): ");
        scanf("%d",&no);
        if (no==1)
            display(head1);
        else
            display(head2);
        break;
    case 5:
        return 0;
    default:
        printf("\nTry again");
        break;
    }
}
}

void insertpoly(struct node **head,int coeff,int power)
{
 New=(struct node*)malloc(sizeof(struct node));
 New->coef=coeff;
 New->pow=power;
 New->next=null;
 if(*head==null)
   *head=New;
 else
 {
  temp=*head;
  while(temp->next!=null)
  {
    temp=temp->next;
  }
  temp->next=New;
 }
}

void addpoly(struct node *h1,struct node *h2)
```

```
{
   head3=null;
   temp1=h1;
   temp2=h2;
   while(temp1!=null && temp2!=null)
   {
      if(temp1->pow>temp2->pow)
      {
         insertpoly(&head3,temp1->coef,temp1->pow);
         temp1=temp1->next;
      }
      else if(temp1->pow<temp2->pow)
      {
         insertpoly(&head3,temp2->coef,temp2->pow);
         temp2=temp2->next;
      }
      else
      {
         insertpoly(&head3,temp1->coef+temp2->coef,temp1->pow);
         temp1=temp1->next;
         temp2=temp2->next;
      }
   }
   while(temp1!=null)
   {
      insertpoly(&head3,temp1->coef,temp1->pow);
      temp1=temp1->next;
   }
   while(temp2!=null)
   {
      insertpoly(&head3,temp2->coef,temp2->pow);
      temp2=temp2->next;
   }
   display(head3);
}

void subpoly(struct node *h1,struct node *h2)
{
   head3=null;
   temp1=h1;
   temp2=h2;
   while(temp1!=null && temp2!=null)
   {
      if(temp1->pow>temp2->pow)
```

```
        {
            insertpoly(&head3,temp1->coef,temp1->pow);
            temp1=temp1->next;
        }
        else if(temp1->pow<temp2->pow)
        {
            insertpoly(&head3,-temp2->coef,temp2->pow);
            temp2=temp2->next;
        }
        else
        {
            insertpoly(&head3,temp1->coef-temp2->coef,temp1->pow);
            temp1=temp1->next;
            temp2=temp2->next;
        }
    }
    while(temp1!=null)
    {
        insertpoly(&head3,temp1->coef,temp1->pow);
        temp1=temp1->next;
    }
    while(temp2!=null)
    {
        insertpoly(&head3,-temp2->coef,temp2->pow);
        temp2=temp2->next;
    }
    display(head3);
}

void multiplypoly(struct node *h1,struct node *h2)
{
    head3=null;
    temp1=h1;
    temp2=h2;
    while(temp1!=null)
    {

        temp2=h2;
        while(temp2!=null)
        {
            insertpoly(&head3,temp1->coef*temp2->coef,temp1->pow+temp2->pow);
            temp2=temp2->next;
        }
        temp1=temp1->next;
```

```
    }
    temp=head3;
    while(temp!=null)
    {
      n=temp->next;
      curr=temp;
      while(n!=null)
      {
        if(temp->pow==n->pow)
        {
          temp->coef=temp->coef+n->coef;
          curr->next=n->next;
          n=n->next;
        }
        else
        {
          n=n->next;
        }
        curr=curr->next;
      }
      temp=temp->next;
    }
    display(head3);
}

void display(struct node *head)
{
    temp=head;
    printf("\nThe Result: ");
    while(temp->next!=null)
    {
      printf("(%dx^%d)+",temp->coef,temp->pow);
      temp=temp->next;
    }
    printf("(%dx^%d)",temp->coef,temp->pow);
}
```

OUTPUT:

```
                    POLYNOMIAL MANIPULATION
        First expression
Enter the number of terms in the first list:3

Enter the coefficient of 1 term:4
```

```
Enter the power of the 1 term:3

Enter the coefficient of 2 term:5

Enter the power of the 2 term:2

Enter the coefficient of 3 term:6

Enter the power of the 3 term:0

                Second expression
Enter the number of terms in the second list:3

Enter the coefficient of 1 term:3

Enter the power of the 1 term:2

Enter the coefficient of 2 term:8

Enter the power of the 2 term:1

Enter the coefficient of 3 term:4

Enter the power of the 3 term:0

  MENU:-
        1.addition
        2.subtraction
        3.multiplication
        4.display
        5.exit
Enter your choice:1

The Result: (4x^3)+(8x^2)+(8x^1)+(10x^0)

    MENU:-
        1.addition
        2.subtraction
        3.multiplication
        4.display
        5.exit
Enter your choice:2

The Result: (4x^3)+(2x^2)+(-8x^1)+(2x^0)
```

```
    MENU:-
       1.addition
       2.subtraction
       3.multiplication
       4.display
       5.exit
Enter your choice:3

The Result: (12x^5)+(47x^4)+(56x^3)+(38x^2)+(48x^1)+(24x^0)

       MENU:-
       1.addition
       2.subtraction
       3.multiplication
       4.display
       5.exit
Enter your choice:4

Enter the list you want to display (1/2): 1

The Result: (4x^3)+(5x^2)+(6x^0)

  MENU:-
       1.addition
       2.subtraction
       3.multiplication
       4.display
       5.exit
Enter your choice:4
 Enter the list you want to display (1/2): 2
The Result: (3x^2)+(8x^1)+(4x^0)

  MENU:-
       1.addition
       2.subtraction
       3.multiplication
       4.display
       5.exit
Enter your choice:5
```

RESULT:

Thus the C program to perform polynomial manipulation was written successfully and output is verified.

EX.NO : 4	SPARSE MATRIX OPERATIONS
DATE :	

AIM:

To write a C Program to perform sparse matrix operations using dynamic implementation.

ALGORITHM:

Step 1 - Start

Step 2 - Read the row and column of (m,n) first and second matrix (p,q)

Step 3 - IF ((n!=p)||(m!=q)) , print("Operations cannot be performed") . ELSE perform

i) Read element of first and second matrix.
ii) Using while loop, performing values operations in matrix repeatedly until return statement is executed.
iii) Read choice and use switch case to select what operations to be performed.

Step 4 - Stop

Insert()

Step 1- Start

Step 2 - Create New node

Step 3 - New->row=rows, New->column=columns, New->val=value, New->next=null

Step 4 - IF (*head==null) set *head=New . ELSE

temp=*head

while(temp->next!=null)

temp=temp->next

temp->next=New

Step 5 - Stop

Addsparsematrix()

Step 1 - Start

Step 2 - head3=null, temp1=h1, temp2=h2

Step 3 - WHILE (temp1!=null && temp2!=null)

i) IF (temp1→ row==temp2→row)

IF (temp1→column==temp2→column)

Insert(&head3,temp1→row,temp1→column, temp1→val +temp2→val)

temp1=temp1→next , temp2=temp2→next

ELSEIF (temp1→column < temp2→column)

Insert(&head3,temp1→row,temp1→column, temp1→val)

temp1=temp1→next

ELSE

Insert(&head3,temp1→row,temp2→column, temp2→val)

temp2=temp1→next

ii) ELSEIF (temp1→row<temp2→row)

Insert (&head3,temp1→row,temp1→column, temp1→val)

temp1=temp1→next

iii) ELSE

Insert (&head3,temp2→row,temp2→column, temp2→val)

temp2=temp2→next

Step 4 – WHILE (temp1!=null)

Insert (&head3,temp1→row,temp1→column, temp1→val)

temp1=temp1→next

Step 5 – WHILE (temp1!=null)

Insert (&head3,temp2→row,temp2→column, temp2→val)

temp2=temp2→next

Step 6 – Stop

Subsparsematrix()

Step 1 - Start

Step 2 - head3=null, temp1=h1, temp2=h2

Step 3 - WHILE (temp1!=null && temp2!=null)

i) IF (temp1→ row==temp2→row)

IF (temp1→column==temp2→column)

Insert(&head3,temp1→row,temp1→column, temp1→val - temp2→val)

temp1=temp1→next , temp2=temp2→next

ELSEIF (temp1→column < temp2→column)

Insert(&head3,temp1→row,temp1→column, temp1→val)

temp1=temp1→next

ELSE

Insert(&head3,temp1→row,temp2→column, temp2→val)

temp2=temp1→next

ii) ELSEIF (temp1→row<temp2→row)

Insert (&head3,temp1→row,temp1→column, temp1→val)

temp1=temp1→next

iii) ELSE

Insert (&head3,temp2→row,temp2→column, temp2→val)

temp2=temp2→next

Step 4 – WHILE (temp1!=null)

Insert (&head3,temp1→row,temp1→column, temp1→val)

temp1=temp1→next

Step 5 – WHILE (temp1!=null)

Insert (&head3,temp2→row,temp2→column, temp2→val)

temp2=temp2→next

Step 6 - Stop

Transpose()

Step 1- Start

Step 2- head3=null, temp=h

Step 3- WHILE (temp1!=null)

insert (&head3,temp1→column,temp1→row,temp1→val)

temp1=temp1→next

Step 4- Stop

Display()

Step 1- Start

Step 2- temp=head3

Step 3- WHILE (temp!=null)

Print (temp→row,temp→column,temp→val)

temp =temp→next

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define null 0
int i,j,a,b,c[50][50],d[50][50],rows,columns,values,y,ch,n,m,p,q,no;
struct node
```

```
{
  int row,column,val;
  struct node *next;
}*curr,*head1=null,*head2=null,*head3=null,*temp,*temp1,*temp2,*New;

void insert(struct node **,int,int,int);
void addsparsematrix(struct node *h1,struct node *h2);
void subsparsematrix(struct node *h1,struct node *h2);
void transpose(struct node *);
void display(struct node *);


int main()
{
 printf("\n\t\tSPARSE MATRIX MANIPULATION");
 printf("\nEnter the rows of first matrix:");
 scanf("%d",&m);
 printf("\nEnter the columns of first matrix:");
 scanf("%d",&n);
 printf("\nEnter the rows of second matrix:");
 scanf("%d",&p);
 printf("\nEnter the columns of second matrix:");
 scanf("%d",&q);
 if((m!=p) || (n!=q))
 {
  printf("\n\tMATRIX MANIPULATION NOT POSSIBLE");
 }
 else
 {
  printf("\n\nEnter the elements of the first matrix:\n");
  for(i=0;i<m;i++)
  {
     for(j=0;j<n;j++)
     {
        scanf("%d",&c[i][j]);
        if(c[i][j]!=0)
        {
          insert(&head1,i,j,c[i][j]);
        }
     }
  }
  printf("\n\nEnter the elements of the second matrix:\n");
  for(i=0;i<p;i++)
  {
```

```
    for(j=0;j<q;j++)
    {
       scanf("%d",&d[i][j]);
       if(d[i][j]!=0)
       {
        insert(&head2,i,j,d[i][j]);
       }
    }
}
while(1)
{
    printf("\n\n\tMENU:-");
    printf("\n\t1.Addition\n\t2.Subtraction\n\t3.Transpose\n\t4.Display\n\t5.Exit");
    printf("\nEnter your choice:");
    scanf("%d",&ch);
    switch(ch)
    {
    case 1:
       addsparsematrix(head1,head2);
       break;
    case 2:
       subsparsematrix(head1,head2);
       break;
    case 3:
       printf("\nEnter which matrix you need to be transpose(1/2) : ");
       scanf("%d",&y);
       switch(y)
       {
       case 1:
          transpose(head1);
          break;
       case 2:
          transpose(head2);
          break;
       default:
          printf("\n***INVALID LIST***");
       }
       break;
    case 4:
       printf("\n Enter the matrix you want to display (1/2): ");
       scanf("%d",&no);
       if (no==1)
          display(head1);
       else
```

```
                display(head2);
            break;
        case 5:
            return 0;
        default:
            printf("\nTry again");
            break;
        }
    }
  }
}


void insert(struct node **head,int rows,int columns,int values)
{
    New=(struct node*)malloc(sizeof(struct node));
    New->row=rows;
    New->column=columns;
    New->val=values;
    New->next=null;
    if(*head==null)
        *head=New;
    else
    {
        temp=*head;
        while(temp->next!=null)
        {
            temp=temp->next;
        }
        temp->next=New;
    }
}


void addsparsematrix(struct node *h1,struct node *h2)
{
    head3=null;
    temp1=h1;
    temp2=h2;
    while(temp1!=null && temp2!=null)
    {
        if(temp1->row==temp2->row)
        {
            if(temp1->column==temp2->column)
```

```
            {
               insert(&head3,temp1->row,temp1->column,temp1->val+temp2->val);
               temp1=temp1->next;
               temp2=temp2->next;
            }
            else if(temp1->column<temp2->column)
            {
               insert(&head3,temp1->row,temp->column,temp1->val);
               temp1=temp1->next;
            }
            else
            {
               insert(&head3,temp2->row,temp2->column,temp2->val);
               temp2=temp2->next;
            }
         }
         else if(temp1->row<temp2->row)
         {
            insert(&head3,temp1->row,temp1->column,temp1->val);
            temp1=temp1->next;
         }
         else
         {
            insert(&head3,temp2->row,temp2->column,temp2->val);
            temp2=temp2->next;
         }
      }
      while(temp1!=null)
      {
         insert(&head3,temp1->row,temp->column,temp1->val);
         temp1=temp1->next;
      }
      while(temp1!=null)
      {
         insert(&head3,temp2->row,temp2->column,temp2->val);
         temp2=temp2->next;
      }
      display(head3);
}

void subsparsematrix(struct node *h1,struct node *h2)
{
      head3=null;
      temp1=h1;
```

```
temp2=h2;
while(temp1!=null && temp2!=null)
{
   if(temp1->row==temp2->row)
   {
      if(temp1->column==temp2->column)
      {
         insert(&head3,temp1->row,temp1->column,temp1->val-temp2->val);
         temp1=temp1->next;
         temp2=temp2->next;
      }
      else if(temp1->column<temp2->column)
      {
         insert(&head3,temp1->row,temp->column,temp1->val);
         temp1=temp1->next;
      }
      else
      {
         insert(&head3,temp2->row,temp2->column,temp2->val);
         temp2=temp2->next;
      }
   }
   else if(temp1->row<temp2->row)
   {
      insert(&head3,temp1->row,temp1->column,temp1->val);
      temp1=temp1->next;
   }
   else
   {
      insert(&head3,temp2->row,temp2->column,temp2->val);
      temp2=temp2->next;
   }
}
while(temp1!=null)
{
   insert(&head3,temp1->row,temp->column,temp1->val);
   temp1=temp1->next;
}
while(temp1!=null)
{
   insert(&head3,temp2->row,temp2->column,temp2->val);
   temp2=temp2->next;
}
display(head3);
```

```
}


void transpose(struct node *h)
{
   head3=null;
   temp1=h;
   while(temp1!=null)
   {
      insert(&head3,temp1->column,temp1->row,temp1->val);
      temp1=temp1->next;
   }
   display(head3);
}

void display(struct node *head)
{
   temp=head;
   printf("\nRESULT:- \n");
   printf("ROW\tCOLUMN\tVALUE\n");
   while(temp!=null)
   {
      printf("%d\t%d\t%d\n",temp->row,temp->column,temp->val);
      temp=temp->next;
   }
}
```

OUTPUT:

```
  SPARSE MATRIX MANIPULATION

Enter the rows of first matrix:3

Enter the columns of first matrix:3

Enter the rows of second matrix:3

Enter the columns of second matrix:3

Enter the elements of the first matrix:
0   2   5
0   0   3
0   0   2
```

```
Enter the elements of the second matrix:
5  0  0
0  0  2
0  0  6


MENU:-
      1.Addition
      2.Subtraction
      3.Transpose
      4.Display
      5.Exit
Enter your choice:4

Enter the matrix you want to display (1/2): 1

RESULT:-
ROW    COLUMN  VALUE
0              1          2
0              2          5
1              2          3
2              2          2


MENU:-
      1.Addition
      2.Subtraction
      3.Transpose
      4.Display
      5.Exit
Enter your choice:4

Enter the matrix you want to display (1/2): 2


RESULT:-
ROW    COLUMN  VALUE
0              0          5
1              2          2
2              2          6


MENU:-
      1.Addition
```

```
        2.Subtraction
        3.Transpose
        4.Display
        5.Exit
Enter your choice:1



RESULT:-
ROW     COLUMN  VALUE
0               0               5
0               1               2
0               2               5
1               2               5
2               2               8


MENU:-
        1.Addition
        2.Subtraction
        3.Transpose
        4.Display
        5.Exit
Enter your choice:2

RESULT:-
ROW     COLUMN  VALUE
0               0               5
0               1               2
0               2               5
1               2               1
2               2               -4


MENU:-
        1.Addition
        2.Subtraction
        3.Transpose
        4.Display
        5.Exit
Enter your choice:3

Enter which matrix you need to be transpose(1/2) : 1
```

RESULT:-

ROW	COLUMN	VALUE
1	0	2
2	0	5
2	1	3
2	2	2

MENU:-

1.Addition
2.Subtraction
3.Transpose
4.Display
5.Exit

Enter your choice:3

Enter which matrix you need to be transpose(1/2) : 2

RESULT:-

ROW	COLUMN	VALUE
0	0	5
2	1	2
2	2	6

MENU:-

1.Addition
2.Subtraction
3.Transpose
4.Display
5.Exit

Enter your choice:5

RESULT:

Thus the C Program to perform sparse matrix operations using dynamic implementation was written successfully and output is verified.

EX.NO : 5	CONVERTING INFIX TO POSTFIX EXPRESSIONS
DATE :	

AIM:

To write a C Program for converting infix expression to postfix expression.

ALGORITHM:

Step 1 - If the character is an operand, place it onto the output.

Step 2 - If the character is an operator, push it onto the stack, if the stack operator has higher or equal priority than the input operator, then pop the operator from the stack to the output and place the operator from the input onto the stack.

Step 3 – If the character is right parenthesis, pop all the operators from the stack till left parenthesis is encountered. Discard both the parenthesis in the output.

Step 4 - If the character is left parenthesis, push it onto the stack.

Step 5 – If the character is #, then pop all the content into output and make stack empty.

PROGRAM:

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top=-1,top;
void push(char X)
{
   stack[++top]=X;
}
char pop()
{
   if(top==-1)
      return -1;
   else
      return stack[top--];
}
int priority(char X)
{
   if(X=='(')
      return 0;


   if(X=='+'||X=='-')
      return 1;
   if(X=='^'||X=='/'||X=='*'||X=='%')
      return 2;
   return 0;
}
int main()
{
   printf("\n\t\t INFIX TO POSTFIX EXPRESSION ");
```

```
    char exp[100];
    char X,Y;
    printf("\n Enter the expression:");
    scanf("%s",&exp);
    printf("\n");
    int i=0;
    while(X=exp[i])
    {
       if(isalnum(X))
          printf("%c",X);
       else if(X=='(')
          push(X);
       else if(X==')')
       {
          while((Y=pop())!='(')
                printf("%c" ,Y);
       }
       else
       {
          while(priority(stack[top])>=priority(X))
             printf("%c",pop());
          push(X);
       }
       i++;
    }
    while(top!=-1)
       printf("%c",pop());
}
```

OUTPUT:

```
INFIX TO POSTFIX EXPRESSION
Enter the expression: A+(B*C-(D/E^F)*E)*H
ABC*DE/F^E*-H*+
```

RESULT:

Thus the C program for converting infix to postfix expression is executed and output is verified successfully.

EX.NO : 6	EVALUATING POSTFIX EXPRESSIONS
DATE :	

AIM:

To write a C Program for evaluating postfix expressions.

ALGORITHM:

Read the given postfix expressions, one character at a time until it encounters the delimiter(#).

Step 1 - If the character is an operand, push it's associated value on to the stack.

Step 2 - If the character is an operator, pop two values from the stack, apply the operator to them and push the result back to the stack.

PROGRAM:

```
#include<stdio.h>
#include<ctype.h>
int stack[20];
int top=-1;
void push(int x)
{
   top=top+1;
   stack[top]=x;
}
int pop()
{
   return stack[top--];
}
int main()
{
   printf("\n\t\t EVALUATING POSTFIX EXPRESSION ");
   char exp[20];
   char z;
   int n1,n2,n3,num,i=0;
   printf("\n Enter the expression:");
   scanf("%s",&exp);
   while(z=exp[i])
   {
      if(isdigit(z))
      {
         num=z-48;
         push(num);
      }
      else
      {
         n1=pop();
```

```
                n2=pop();
                switch(z)
                {
                case '+':
                   {
                      n3=n1+n2;
                      break;
                   }
                case '-':
                   {
                      n3=n2-n1;
                      break;
                   }
                case '*':
                   {
                      n3=n1*n2;
                      break;
                   }
                case '/':
                   {
                      n3=n2/n1;
                      break;
                   }
                }
                push(n3);
             }
             i++;
          }
          printf("\nThe result of expression %s=%d",exp,pop());
       }
```

OUTPUT:

EVALUATING POSTFIX EXPRESSION

Enter the postfix expression:2536+**5/2-

The result of postfix expression 2536+**5/2- = 16

RESULT:

Thus the C program for evaluating postfix expression is executed and output is verified successfully.

EX.NO : 7A	BINARY TREE TRAVERSALS – RECURSIVE FUNCTIONS
DATE :	

AIM:

To write a C Program to perform various Binary Tree Traversal using recursive function.

ALGORITHM:

Step 1 - Start

Step 2 - Include all the header files which are used in the program and define a struct node with one data field (int data) and two pointer field (struct node *left,*right).

Step 3 - Declare all the recursive functions used in Binary Tree Traversal.

Step 4 - In main method, read number element in tree and insert the element in a tree and make suitable function calls to perform binary tree traversal.

Step 5 – Stop

INSERT()

Step 1 – Check whether the tree is empty. (T==null)

Step 2- If it yes, then create New node and set

Step 2.1 – New->data=X

Step 2.2 – New->left=null

Step 2.3 – New->right=null

Step 2.4 – T=New

Step 3 – Else check if(X<T->data), yes then T->left=insert(X,T->left), else if(X>T->data),yes then T->right=insert(X,T->right), else display “Element is already exist in tree”.

INORDER()

Step 1 – Check whether tree is not empty.(T!=null)

Step 2 – If yes, then inorder(T->left), display (T->data), inorder(T->right).

PREORDER()

Step 1 – Check whether tree is not empty.(T!=null)

Step 2 – If yes, then display (T->data), preorder(T->left), preorder(T->right).

POSTORDER()

Step 1 – Check whether tree is not empty.(T!=null)

Step 2 – If yes, then postorder(T->left), postorder(T->right), display (T->data).

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define null 0
int n,a;
struct node
{
int data;
struct node *left,*right;
}*temp,*New,*T1;
void insert(int,struct node **T);
void inorder(struct node *t);
void preorder(struct node *t);
void postorder(struct node *t);
int main()
{
   printf("\t*****BINARY TREE TRAVERSAL - RECURSIVE FUNCTION*****");
   printf("\n\nenter the number of element in the tree:");
   scanf("%d",&n);
   for(int i=1;i<=n;i++)
   {
      printf("\nenter the %d element:",i);
      scanf("%d",&a);
      insert(a,&T1);
   }
   printf("\ninorder traversal:");
   inorder(T1);
   printf("\n\npreorder traversal:");
   preorder(T1);
```

```
    printf("\n\npostorder traversal:");
    postorder(T1);
}
void insert(int x,struct node **T)
{
    if(*T==null)
    {
        New=(struct node*)malloc(sizeof(struct node));
        New->data=x;
        New->left=null;
        New->right=null;
        *T=New;
    }
    else
    {
        temp=*T;
        if(x<temp->data)
        {
            insert(x,&temp->left);
        }
        else if(x>temp->data)
        {
            insert(x,&temp->right);
        }
        else
        {
            printf("\nelement is already exist in the tree\n");
        }
    }
}
void inorder(struct node *t)
{
    if(t!=null)
    {
        inorder(t->left);
        printf("%d\t",t->data);
        inorder(t->right);
    }
}
void preorder(struct node *t)
{
    if(t!=null)
    {
        printf("%d\t",t->data);
```

```
            preorder(t->left);
            preorder(t->right);
        }
    }
    void postorder(struct node *t)
    {
        if(t!=null)
        {
            postorder(t->left);
            postorder(t->right);
            printf("%d\t",t->data);
        }
    }
```

OUTPUT:

```
*****BINARY TREE TRAVERSAL - RECURSIVE FUNCTION******

enter the number of element in the tree:5

enter the 1 element:75

enter the 2 element:66

enter the 3 element:33

enter the 4 element:98

enter the 5 element:12

inorder traversal:12    33      66      75      98

preorder traversal:75   66      33      12      98

postorder traversal:12  33      66      98      75
```

RESULT:

Thus the C program to perform various Binary Tree Traversal using recursive function was written and output is verified successfully.

EX.NO : 7B	BINARY TREE TRAVERSALS – NON-RECURSIVE FUNCTIONS
DATE :	

AIM:

To write a C Program to perform various Binary Tree Traversal using non-recursive function.

ALGORITHM:

Step 1 - Start

Step 2 - Include all the header files which are used in the program and define a struct node with one data field (int data) and two pointer field (struct node *left,*right).

Step 3 – Declare dynamic stack to perform binary tree traversal using non-recursive function.

Step 4 - Declare all the functions used in Binary Tree Traversal.

Step 5 - In main method, read number element in tree and insert the element in a tree and make suitable function calls to perform binary tree traversal.

Step 6 – stop

PUSH()

Step 1 – Create New node and set

Step 1.1 – New->data=t->data

Step 1.2 – New->left=t->left

Step 1.3 – New->right=t->right

Step 1.4 – New->next=null

Step 2 – Check if(top==null) set top=New else set New->next = top and top=New.

POP()

Step 1 – check if(top==null) then display “stack is empty” else set top=top->next and return top.

EMPTY()

Step 1 – Check if(top==null) then return 1 else return 0.

INSERT()

Step 1 – Check whether the tree is empty. (T==null)

Step 2- If it yes, then create New node and set

Step 2.1 – New->data=X

Step 2.2 – New->left=null

Step 2.3 – New->right=null

Step 2.4 – T=New

Step 3 – Else check if(X<T->data), yes then T->left=insert(X,T->left), else if(X>T->data),yes then T->right=insert(X,T->right), else display “Element is already exist in tree”.

INORDER()

Step 1 – Repeat the step 2 and 3 while (t!=null)

Step 2 – push(top1,t)

Step 3 – t=t->left

Step 4 – Repeat the step 5 to 7 while (!empty(top1))

Step 5 – t=pop(top1)

Step 6 – display “t->data”

Step 7 – t=t->right

Step 8 – Repeat the step 9 and 10 while(t!=null)

Step 9 – push(top1,t)

Step 10 – t=t->left

PREORDER()

Step 1 – push(top1,t)

Step 2 – Repeat the step 3 to 6 while(!empty(top1))

Step 3 – t=pop(top1)

Step 4 – display " t->data "

Step 5 – Check if(t->right!=null), yes then do push(top1,t->right).

Step 6 – Check if(t->left!=null), yes then do push(top1,t->left).

POSTORDER()

Step 1 – push(top1,t)

Step 2 – Repeat the step 3 to 6 while(!empty(top1))

Step 3 – t=pop(top1)

Step 4 – push(top2,t)

Step 5 – Check if(t->left!=null), yes then do push(top1,t->left).

Step 6 – Check if(t->right!=null), yes then do push(top1,t->right).

Step 7 – Repeat the step 8 and 9 while(!empty(top2))

Step 8 – t=pop(top2)

Step 9 – display " t->data "

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define null 0
int n,a,b;
struct node
{
int data;
struct nodes *left,*right,*next;
}*temp,*New,*c,*T1,*top1=null,*top2=null;
void push(struct node **top,struct node *t);
struct node* pop(struct node **top);
int empty(struct node *top);
void insert(int,struct node **T);
void inorder(struct node *t);
void preoder(struct node *t);
```

```
void postorder(struct node *t);
int main()
{
   printf("\t*****BINARY TREE TRAVERSAL NON-RECURSIVE FUNCTION*****");
   printf("\n\nenter the number of element in the tree:");
   scanf("%d",&n);
   for(int i=1;i<=n;i++)
   {
      printf("\nenter the %d element:",i);
      scanf("%d",&a);
      insert(a,&T1);
   }
   printf("\ninorder traversal:");
   inorder(T1);
   printf("\n\npreorder traversal:");
   preorder(T1);
   printf("\n\npostorder traversal:");
   postorder(T1);
}
void push(struct node **top,struct node *t)
{
   New=(struct node*)malloc(sizeof(struct node));
   New->data=t->data;
   New->left=t->left;
   New->right=t->right;
   New->next=null;
   if(top==null)
      *top=New;
   else
   {
      New->next=*top;
      *top=New;
   }
}
struct node* pop(struct node **top)
{
   if(*top==null)
   {
      printf("\nstack is empty");
   }
   else
   {
      c=*top;
      *top=c->next;
```

```
        return c;
    }
}
int empty(struct node *top)
{
    if(top==null)
        return 1;
    else
        return 0;
}
void insert(int x,struct node **T)
{
    if(*T==null)
    {
        New=(struct node*)malloc(sizeof(struct node));
        New->data=x;
        New->left=null;
        New->right=null;
        *T=New;
    }
    else
    {
        temp=*T;
        if(x<temp->data)
        {
            insert(x,&temp->left);
        }
        else if(x>temp->data)
        {
            insert(x,&temp->right);
        }
        else
        {
            printf("\nelement is already exist in the tree\n");
        }
    }
}
void inorder(struct node *t)
{
    top1=null;
    while(t!=null)
    {
        push(&top1,t);
        t=t->left;
```

```
    }
    while(!empty(top1))
    {
        t=pop(&top1);
        printf("%d\t",t->data);
        t=t->right;
        while(t!=null)
        {
            push(&top1,t);
            t=t->left;
        }
    }
}
void preorder(struct node *t)
{
    top1=null;
    push(&top1,t);
    while(!empty(top1))
    {
        t=pop(&top1);
        printf("%d\t",t->data);
        if(t->right!=null)
            push(&top1,t->right);
        if(t->left!=null)
            push(&top1,t->left);
    }
}
void postorder(struct node *t)
{
    top1=null;
    top2=null;
    push(&top1,t);
    while(!empty(top1))
    {
        t=pop(&top1);
        push(&top2,t);
        if(t->left!=null)
            push(&top1,t->left);
        if(t->right!=null)
            push(&top1,t->right);
    }
    while(!empty(top2))
    {
        t=pop(&top2);
```

```
            printf("%d\t",t->data);
        }
    }
```

OUTPUT:

```
        *****BINARY TREE TRAVERSAL NON-RECURSIVE FUNCTION*****

    enter the number of element in the tree:5

    enter the 1 element:67

    enter the 2 element:9

    enter the 3 element:23

    enter the 4 element:64

    enter the 5 element:95

    inorder traversal:9     23      64      67      95

    preorder traversal:67   9       23      64      95

    postorder traversal:64  23      9       95      67
```

RESULT:

Thus the C program to perform various Binary Tree Traversal using non-recursive function was written and output is verified successfully.

EX.NO : 8	IMPLEMENTATION OF BINARY SEARCH TREES
DATE :	

AIM:

To write a C Program to implement various binary search tree operations.

ALGORITHM:

Step 1 - Start

Step 2 – Read number of element in tree and read a element in tree and insert it.

Step 3 – Using while loop perform the operation in binary search tree until return statement is executed.

Step 4 – Read choice.

Step 5 – Switch case is used to select which operations to be perform and perform according to the case.

Step 6 – Stop

CREATE()

Step 1 – Start

Step 2 – for(i=0;i<n;i++) do Read element of tree and insert(a,T).

Step 3 – Stop

INSERT()

Step 1 – Start

Step 2 – if(T==null) do create New Node, New->data=X, New->left=null, New->right=null , T=New.

Step 3 – else do temp=T, if(X<temp->data) do insert(X,temp->left) else if(X>temp->data) do insert(X,temp->right) else print(“element already exist”).

Step 4 – Stop

FIND()

Step 1 – Start

Step 2 – if(T==null) do print (“element not found”).

Step 3 – else do temp=T, if(X<temp->data) do find(X,temp->left) else if(X>temp->data) do find(X,temp->right) else print(“element already exist”).

Step 4 – Stop

FINDMIN()

Step 1 – Start

Step 2 – if(T==null) return null else if(T->left==null) return T->data else return FindMin(T->left).

Step 3 – Stop

FINDMAX()

Step 1 – Start

Step 2 – if(T==null) return null else if(T->right==null) return T->data else return FindMax(T->right).

Step 3 – Stop

MAKEEMPTY()

Step 1 – Start

Step 2 – if(T!=null) do T->left=MakeEmpty(T->left), T->right=MakeEmpty(T->right), free(T).

Step 3 – Return null

DELETE()

Step 1 – Start

Step 2 – if(T==null) do print (“element not found”) else if(X<T->data) do delete(X,T->left) else if(X>T->data) do delete(X,T->right) else do

Step 2.1 – if(T->left!=null && T->right!=null) do T->data =FindMin(T->right), T->right=delete(T->data,T->right) else do if(T->left==null) do T=T->right else if(T->right==null) do T=T->left.

Step 3 – return T

DISPLAY()

Step 1- Start

Step 2 – if(T!=null) do display(T->left) , print "T->data ", display(T->right)

Step 3 - Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define null 0
int n,a,ch;
struct node
{
   int data;
   struct node *left,*right;
}*temp,*New,*T1;
void create(int);
void insert(int,struct node **T);
struct node *Delete(int,struct node *T);
struct node *FindMin(struct node *T);
struct node *FindMax(struct node *T);
struct node *MakeEmpty(struct node *T);
void display(struct node *T);
int main()
{
   printf("BINARY SEARCH TREE");
   printf("\nenter the number of element in the tree:");
   scanf("%d",&n);
   create(n);
   while(1)
   {
      printf("\nMENU\n\t1.insert\n\t2.delete\n\t3.find\n\t4.findmin\n\t5.findmax\n\t6.make
empty\n\t7.display\n\t8.exit");
      printf("\nenter your choice:");
```

```
scanf("%d",&ch);
switch(ch)
{
   case 1:
   {
      printf("\nenter the element to be insert:");
      scanf("%d",&a);
      insert(a,&T1);
      break;
   }
   case 2:
   {
      if(T1==null)
      printf("\ntree is empty");
      else
      {
         printf("\nenter the element to be delete:");
         scanf("%d",&a);
         T1=Delete(a,T1);
      }
      break;
   }
   case 3:
   {
      printf("\nenter the element to be search:");
      scanf("%d",&a);
      find(a,T1);
      break;
   }
   case 4:
   {
      a=FindMin(T1);
      printf("%d",a);
      break;
   }
   case 5:
   {
      a=FindMax(T1);
      printf("%d",a);
      break;
   }
   case 6:
   {
      T1=MakeEmpty(T1);
```

```
                break;
            }
            case 7:
            {
                if(T1==null)
                    printf("\ntree is empty");
                else
                    display(T1);
                break;
            }
            case 8:
                return 0;
            default:
                printf("\ntry again");
                break;
        }
    }
}
void create(int b)
{
    for(int i=0;i<n;i++)
    {
        printf("\nenter the element of tree:");
        scanf("%d",&a);
        insert(a,&T1);
    }
}
void insert(int x,struct node **T)
{
    if(*T==null)
    {
        New=(struct node*)malloc(sizeof(struct node));
        New->data=x;
        New->left=null;
        New->right=null;
        *T=New;
    }
    else
    {
        temp=*T;
        if(x<temp->data)
        {
            insert(x,&temp->left);
        }
```

```
        else if(x>temp->data)
        {
           insert(x,&temp->right);
        }
        else
        {
           printf("\nelement is already exist in the tree\n");
        }
    }
}
void find(int x,struct node *T)
{
    if(T==null)
    {
       printf("\n%d is not found",x);
    }
    else
    {
       temp=T;
       if(x<temp->data)
       {
          find(x,temp->left);
       }
       else if(x>temp->data)
       {
          find(x,temp->right);
       }
       else
       {
          printf("\n%d is found",x);
       }
    }
}
struct node *FindMin(struct node *T)
{
    if(T==null)
       return null;
    else if(T->left==null)
       return T->data;
    else
       return FindMin(T->left);
}
struct node *FindMax(struct node *T)
{
```

```
    if(T==null)
        return null;
    else if(T->right==null)
        return T->data;
    else
        return FindMax(T->right);
}
struct node *MakeEmpty(struct node *T)
{
    if(T!=null)
    {
        T->left=MakeEmpty(T->left);
        T->right=MakeEmpty(T->right);
        free(T);
        return null;
    }
}
struct node *Delete(int x,struct node *T)
{
    if(T==null)
        printf("\nelement not found");
    else if(x<T->data)
        T->left=Delete(x,T->left);
    else if(x>T->data)
        T->right=Delete(x,T->right);
    else
    {
        if(T->left!=null && T->right!=null)
        {
            T->data=FindMin(T->right);
            T->right=Delete(T->data,T->right);
        }
        else
        {
            if(T->left==null)
                T=T->right;
            else if(T->right==null)
                T=T->left;
        }
    }
    return T;
}
void display(struct node *T)
{
```

```
    if(T!=null)
    {
        display(T->left);
        printf("%d\t",T->data);
        display(T->right);
    }
}
```

OUTPUT:

```
BINARY SEARCH TREE
enter the number of element in the tree:5

enter the element of tree:45

enter the element of tree:65

enter the element of tree:84

enter the element of tree:97

enter the element of tree:56

MENU
    1.insert
    2.delete
    3.find
    4.findmin
    5.findmax
    6.make empty
    7.display
    8.exit
enter your choice:7
45    56    65    84    97
MENU
    1.insert
    2.delete
    3.find
    4.findmin
    5.findmax
    6.make empty
    7.display
    8.exit
enter your choice:1
```

```
enter the element to be insert:36

MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:7
36      45      56      65      84      97
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:4
36
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:5
97
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
```

```
        8.exit
enter your choice:3

enter the element to be search:65

65 is found
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:3

enter the element to be search:83

83 is not found
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:2

enter the element to be delete:84

MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:7
36      45      56      65      97
```

```
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:2

enter the element to be delete:45

MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:7
36      56      65      97
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:2

enter the element to be delete:97

MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
```

```
        7.display
        8.exit
enter your choice:7
36      56      65
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:6

MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:7

tree is empty
MENU
        1.insert
        2.delete
        3.find
        4.findmin
        5.findmax
        6.make empty
        7.display
        8.exit
enter your choice:8
```

RESULT:

Thus the C program to implement various Binary Search Tree operation was written successfully and output is verified.

EX.NO : 9	IMPLEMENTATION OF HUFFMAN CODING
DATE :	

AIM:

To write a C program to perform a Huffman tree and find the Huffman coding.

ALGORITHM:

Step 1 – Start

Step 1.1 – Initialise ‘n’ one node trees and label them with the characters of the alphabet.

Step 1.2 – Record the frequency of each character in its tree’s root, to indicate the tree’s weight.

Step 2 – Repeat the following operation until the single tree is obtained.

Step 2.1 – Find two trees with the smallest weight.

Step 2.2 – Make them the left and right subtree of a new tree and record the sum of their weights in the root of new tree as its weight.

Step 3 – The tree constructed by using above algorithm and then find the code that derived from the tree.

Step 4 – Stop

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TREE_HT 50
struct MinHNode
{
  char item;
  unsigned freq;
  struct MinHNode *left, *right;
};
struct MinHeap
{
  unsigned size;
  unsigned capacity;
  struct MinHNode **array;
};
```

```
struct MinHNode *newNode(char item, unsigned freq)
{
struct MinHNode *temp = (struct MinHNode *)malloc(sizeof(struct MinHNode));
temp->left = temp->right = NULL;
temp->item = item;
temp->freq = freq;
return temp;
}
struct MinHeap *createMinH(unsigned capacity)
{
struct MinHeap *minHeap = (struct MinHeap *)malloc(sizeof(struct MinHeap));
minHeap->size = 0;
minHeap->capacity = capacity;
minHeap->array = (struct MinHNode **)malloc(minHeap->capacity * sizeof(struct MinHNode *));
return minHeap;
}
void swapMinHNode(struct MinHNode **a, struct MinHNode **b)
{
struct MinHNode *t = *a;
*a = *b;
*b = t;
}
void minHeapify(struct MinHeap *minHeap, int idx)
{
  int smallest = idx;
  int left = 2 * idx + 1;
  int right = 2 * idx + 2;
  if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)
    smallest = left;
  if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)
    smallest = right;
  if (smallest != idx)
    {
    swapMinHNode(&minHeap->array[smallest], &minHeap->array[idx]);
    minHeapify(minHeap, smallest);
  }
}
int checkSizeOne(struct MinHeap *minHeap)
{
  return (minHeap->size == 1);
}
struct MinHNode *extractMin(struct MinHeap *minHeap)
{
  struct MinHNode *temp = minHeap->array[0];
```

```
  minHeap->array[0] = minHeap->array[minHeap->size - 1];
  --minHeap->size;
  minHeapify(minHeap, 0);
  return temp;
}
void insertMinHeap(struct MinHeap *minHeap, struct MinHNode *minHeapNode)
{
  ++minHeap->size;
  int i = minHeap->size - 1;
  while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq)
 {
    minHeap->array[i] = minHeap->array[(i - 1) / 2];
    i = (i - 1) / 2;
  }
  minHeap->array[i] = minHeapNode;
}
void buildMinHeap(struct MinHeap *minHeap) {
  int n = minHeap->size - 1;
  int i;
  for (i = (n - 1) / 2; i >= 0; --i)
    minHeapify(minHeap, i);
}
int isLeaf(struct MinHNode *root) {
  return !(root->left) && !(root->right);
}
struct MinHeap *createAndBuildMinHeap(char item[], int freq[], int size)
{
  struct MinHeap *minHeap = createMinH(size);
  for (int i = 0; i < size; ++i)
    minHeap->array[i] = newNode(item[i], freq[i]);
  minHeap->size = size;
  buildMinHeap(minHeap);
  return minHeap;
}
struct MinHNode *buildHuffmanTree(char item[], int freq[], int size)
{
  struct MinHNode *left, *right, *top;
  struct MinHeap *minHeap = createAndBuildMinHeap(item, freq, size);
  while (!checkSizeOne(minHeap))
    {
    left = extractMin(minHeap);
    right = extractMin(minHeap);
    top = newNode('$', left->freq + right->freq);
    top->left = left;
```

```
    top->right = right;
    insertMinHeap(minHeap, top);
  }
  return extractMin(minHeap);
}
void printHCodes(struct MinHNode *root, int arr[], int top)
{
  if (root->left)
    {
    arr[top] = 0;
    printHCodes(root->left, arr, top + 1);
  }
  if (root->right)
  {
    arr[top] = 1;
    printHCodes(root->right, arr, top + 1);
  }
  if (isLeaf(root))
  {
    printf("  %c   | ", root->item);
    printArray(arr, top);
  }
}
void HuffmanCodes(char item[], int freq[], int size)
{
  struct MinHNode *root = buildHuffmanTree(item, freq, size);
  int arr[MAX_TREE_HT], top = 0;
  printHCodes(root, arr, top);
}
void printArray(int arr[], int n)
 {
  int i;
  for (i = 0; i < n; ++i)
    printf("%d", arr[i]);
  printf("\n");
}
int main()
{
  char arr[] = {'A', 'B', 'C', 'D'};
  int freq[] = {5, 1, 6, 3};
  int size = sizeof(arr) / sizeof(arr[0]);
  printf(" Char | Huffman code ");
  printf("\n--------------------\n");
  HuffmanCodes(arr, freq, size);
```

```
}
```

OUTPUT:

```
Char | Huffman code
--------------------
 C   | 0
 B   | 100
 D   | 101
 A   | 11
```

RESULT:

Thus the C program to perform a Huffman tree and finding the Huffman coding was written and output is verified successfully**.**

EX.NO : 10	GRAPH REPRESENTATION – ADJACENT MATRIX AND ADJACENCY LISTS
DATE :	

AIM:

To write a C Program to represent the graph using adjacency matrix and adjacency list.

ALGORITHM:

Step 1 – Start.
Step 2 – Read number of vertices.
Step 3 – Using while loop, graph is represented using matrix and list.
Step 4 – Read the choice.
Step 5 – Switch case is used to select which representation is going to perform.
DIRECTEDMATRIX()
Step 1 – Start.
Step 2 – for(i=0;i<a;i++)do for(j=0;j<a;j++)
if(i==j)do x[i][j]
else do read if there is a edge from the vertices(y),x[i][j]=y
Step 3 – Display the matrix by using for loop
Step 4 – Stop.
UNDIRECTEDMATRIX()
Step 1 – Start.
Step 2 – for(i=0;i<a;i++)do for(j=0;j<a;j++)
if(i==j)do x[i][j]=0
Step 3 – for(i=0;i<a;i++)do for(j=0;j<a;j++)do
Read if there is an edge from that vertices,x[i][j]=y,x[j][i]=y
Step 4 – Display matrix by using for loop.
Step 5 – Stop.
INSERT()
Step 1 – Start.
Step 2 – Create New node.
Step 3 – New ->data=X
New ->next=null
Step 4 – if(V==null)do V=New
else do temp=V,
while(temp->next!=null)do temp=temp->next
temp->next=New
Step 5 – Stop.
DISPLAY()
Step 1 – Start.
Step 2 – if(V!=null)
temp=V
while(temp->next !=null)do print(temp->data),temp=temp->next
Step 3 – Stop.
DIRECTEDLIST()
Step 1 – Start.
Step 2 – for(i=0;i<a;i++)do V[i]==null,insert(V[i],i+1)

Step 2.1 – for(j=0;j<a;j++)do if(i!=j)do

Read if there is a edge or not, if(y==1)do insert(V[i],j+1)

Step 3 – Display the graph by using for loop.

Step 4 – Stop.

UNDIRECTEDLIST()

Step 1 – Start.

Step 2 – for(i=0;i<a;i++)do V[i]==null,insert(V[i],i+1)

Step 3 – for(i=0;i<a;i++)do for(j=i+1;j<a;j++)do

Read if there is a edge or not, if(y==1)do insert(V[i],j+1), insert(V[i],j+1)

Step 4 – Display the (list) graph by using for loop.

Step 5 – Stop.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define null 0
int a,n,ch,i,j,k,y;
struct node
  {
     int data;
     struct node *next;
  }*V[50],*temp,*New;
  void directedmatrix();
  void undirectedmatrix();
  void directedlist();
  void undirectedlist();
  void insert(struct node **,int);
  void display(struct node *V);
  int main()
  {
     printf("\t\t\t*****GRAPH REPRESENTATION******");
     printf("\n Enter the number of vertices:");
     scanf("%d",&a);
     while(1)
     {
        printf("\n\n Enter which type of graph:");
        printf("\n\t 1.Directed graph\n\t 2.Undirected graph\n\t 3.Exit");
        printf("\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
           printf("\n\n MENU:\n\t 1.Adjacency matrix\n\t 2.Adjacency list");
           printf("\n Enter your choice :");
           scanf("%d",&n);
           switch(n)
           {
           case 1:
```

```
                directedmatrix();
                break;
            case 2:
                directedlist();
                break;
            default:
                printf("\n Invalid choice");
                break;
            }
            break;
        case 2:
            printf("\n\n MENU:\n\t 1.Adjacency matrix \n\t 2.Adjacency list\n\t 3.Exit");
            printf("\n Enter your choice:");
            scanf("%d",&n);
            switch(n)
            {
            case 1:
                undirectedmatrix();
                break;
            case 2:
                undirectedlist();
                break;
            default:
                printf("\n Invalid choice");
                break;
            }
            break;
        case 3:
            exit(0);
        default:
            printf("Try again");
            }
        }
    }
void directedmatrix()
{
   int x[a][a];
   for(i=0;i<a;i++)
   {
      for(j=0;j<a;j++)
      {
         if(i==j)
            x[i][j]=0;
         else
         {
            printf("Is there an edge from vertex %d to %d:",i+1,j+1);
            scanf("%d",&y);
```

```
                x[i][j]=y;
            }
        }

    }
    printf("\n\n Adjacency matrix for the given directed graph:\n\n");
    for(i=0;i<a;i++)
    {
        for(j=0;j<a;j++)
        {
            printf("%d\t",x[i][j]);
        }
        printf("\n");
    }

}
void undirectedmatrix()
{
    int x[a][a];
    for(i=0;i<a;i++)
    {
        for(j=0;j<a;j++)
        {
            if(i==j)
               x[i][j]=0;

        }
    }

for(i=0;i<a;i++)
{
    for(j=i+1;j<a;j++)
    {
        printf("\n Is there an edge from vertex %d to %d:",i+1,j+1);
        scanf("%d",&y);
        x[i][j]=y;
        x[j][i]=y;
    }
}

printf("\n\n Adjacency matrix for the given undirected graph:\n");
for(i=0;i<a;i++)
{
    for(j=0;j<a;j++)
    {
        printf("%d\t",x[i][j]);
    }
```

```
    printf("\n");
}
}
void insert(struct node **V,int x)
{
    New=(struct node*)malloc(sizeof(struct node));
    New->data=x;
    New->next=null;
    if(*V==null)
        *V=New;
    else
    {
        temp=*V;
        while(temp->next!=null)
        {
            temp=temp->next;
        }
        temp->next=New;
    }
}
void display(struct node *V)
{
    if(V!=null)
    {
        temp=V;
        while(temp->next!=null)
        {
            printf("%d->",temp->data);
            temp=temp->next;
        }
        printf("%d",temp->data);
    }
}
void directedlist()
{
    for(i=0;i<a;i++)
    {
        V[i]=null;
        insert(&V[i],i+1);
        for(j=0;j<a;j++)
        {
            if(i!=j)
            {
                printf("\n Is there an edge from vertex %d to %d:",i+1,j+1);
                scanf("%d",&y);
                if(y==1)
                {
```

```
                    insert(&V[i],j+1);
                }
            }
        }
    }
    printf("\n\n Adjacency list for the given directed graph:");
    for(i=0;i<a;i++)
    {
        printf("\n");
        display(V[i]);
    }
}
void undirectedlist()
{
    for(i=0;i<a;i++)
    {
        V[i]=null;
        insert(&V[i],i+1);
    }
    for(i=0;i<a;i++)
    {
        for(j=i+1;j<a;j++)
        {
            printf("Is there an edge from vertex %d to %d :",i+1,j+1);
            scanf("%d",&y);
            if(y==1)
            {
                insert(&V[i],j+1);
                insert(&V[j],i+1);
            }
        }
    }
        printf("\n\n Adjacency list for the given undirected graph:");
        for(i=0;i<a;i++)
        {
            printf("\n");
            display(V[i]);

        }
}
```

OUTPUT:

```
*****GRAPH REPRESENTATION******
Enter the number of vertices:4
Enter which type of graph:
    1.Directed graph
    2.Undirected graph
    3.Exit
```

```
Enter your choice:1
 MENU:
        1.Adjacency matrix
        2.Adjacency list
 Enter your choice :1
Is there an edge from vertex 1 to 2:1
Is there an edge from vertex 1 to 3:0
Is there an edge from vertex 1 to 4:0
Is there an edge from vertex 2 to 1:0
Is there an edge from vertex 2 to 3:1
Is there an edge from vertex 2 to 4:0
Is there an edge from vertex 3 to 1:1
Is there an edge from vertex 3 to 2:0
Is there an edge from vertex 3 to 4:1
Is there an edge from vertex 4 to 1:0
Is there an edge from vertex 4 to 2:1
Is there an edge from vertex 4 to 3:0
 Adjacency matrix for the given directed graph:
0       1       0       0
0       0       1       0
1       0       0       1
0       1       0       0
 Enter which type of graph:
        1.Directed graph
        2.Undirected graph
        3.Exit
Enter your choice:2
 MENU:
        1.Adjacency matrix
        2.Adjacency list
        3.Exit
 Enter your choice:1
 Is there an edge from vertex 1 to 2:1
 Is there an edge from vertex 1 to 3:1
 Is there an edge from vertex 1 to 4:0
 Is there an edge from vertex 2 to 3:1
 Is there an edge from vertex 2 to 4:1
 Is there an edge from vertex 3 to 4:1
 Adjacency matrix for the given undirected graph:
0       1       1       0
1       0       1       1
1       1       0       1
0       1       1       0
 Enter which type of graph:
        1.Directed graph
        2.Undirected graph
        3.Exit
```

```
Enter your choice:1
 MENU:
        1.Adjacency matrix
        2.Adjacency list
 Enter your choice :2
 Is there an edge from vertex 1 to 2:1
 Is there an edge from vertex 1 to 3:0
 Is there an edge from vertex 1 to 4:0
 Is there an edge from vertex 2 to 1:0
 Is there an edge from vertex 2 to 3:1
 Is there an edge from vertex 2 to 4:0
 Is there an edge from vertex 3 to 1:1
 Is there an edge from vertex 3 to 2:0
 Is there an edge from vertex 3 to 4:1
 Is there an edge from vertex 4 to 1:0
 Is there an edge from vertex 4 to 2:1
 Is there an edge from vertex 4 to 3:0
 Adjacency list for the given directed graph:
1->2
2->3
3->1->4
4->2
 Enter which type of graph:
        1.Directed graph
        2.Undirected graph
        3.Exit
Enter your choice:2
 MENU:
        1.Adjacency matrix
        2.Adjacency list
        3.Exit
 Enter your choice:2
Is there an edge from vertex 1 to 2 :1
Is there an edge from vertex 1 to 3 :1
Is there an edge from vertex 1 to 4 :0
Is there an edge from vertex 2 to 3 :1
Is there an edge from vertex 2 to 4 :1
Is there an edge from vertex 3 to 4 :1
 Adjacency list for the given undirected graph:
1->2->3
2->1->3->4
3->1->2->4
4->2->3
```

RESULT:

Thus the C program to represent the graph using adjacent matrix and adjacent list is written and output is verified successfully.

EX.NO : 11	IMPLEMENTATION OF PRIM'S ALGORITHM
DATE :	

AIM:

To write a C program to find the minimum cost spanning tree by using prim's algorithm.

ALGORITHM:

Step 1 – Start

Step 2 – Read number of node.

Step 3 – Read the adjacency matrix by using for loop.

Step 4 – set visited[i]=1

Step 5 – while(ne<n) do

Step 5.1 - for(i=1,min=999;i<=n;i++) do for(j=1;j<=n;j++) do if(cost[i][j]<min) do if(visited[i]!=0) set min=cost[i][j], a=u=i, b=v=j.

Step 5.2 – if(visited[u]==0 || visited[v]==0) do display(edge,cost), set mincost+=min, visited[b]=1.

Step 5.3 - cost[a][b]=cost[b][a]=999

Step 6 – display mincost.

Step 7 - stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
  {
    printf("\n Enter the number of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
        {
```

```
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                cost[i][j]=999;
            }
        visited[1]=1;
        printf("\n");
        while(ne<n)
        {
            for(i=1,min=999;i<=n;i++)
                for(j=1;j<=n;j++)
                    if(cost[i][j]<min)
                        if(visited[i]!=0)
                            {
                                min=cost[i][j];
                                a=u=i;
                                b=v=j;
                            }
            if(visited[u]==0||visited[v]==0)
            {
                printf("\n Edge %d:(%d %d)cost:%d",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\nMinimum cost=%d",mincost);
        getch();
    }
```

OUTPUT:

Enter the number of nodes:4

Enter the adjacency matrix:

0
5
1
6
5
0
7
3
1
7
0

2
6
3
2
0

Edge 1:(1 3)cost:1
Edge 2:(3 4)cost:2
Edge 3:(4 2)cost:3
Minimum cost=6

RESULT:

Thus the C program to find the minimum cost spanning tree by using prim's algorithm was written and output is verified successfully**.**

EX.NO : 12A	SHORTEST PATHS – DIJKSTRA'S ALGORITHM
DATE :	

AIM:

To write a C program to find the shortest path to all the other vertices by using Dijkstra's algorithm.

ALGORITHM:

Step 1 – Start

Step 2 – Read number of vertices.

Step 3 – get()

Step 4 – Read source & destination.

Step 5 – Shortest path -> call shrt(src,dest)

Step 6 – Stop

GET()

Step 1 – Start

Step 2 – Read the edge weight of the given vertices by using nested for loop.

Step 3 – Stop

SHRT()

Step 1 – start

Step 2 – for(=1;i<=n;++) do perm[i]=0, dist[i]=infinity.

Step 3 – Set perm[src]=1, dist[src]=0, current=src.

Step 4 – while(current!=dest) do small=infinity, start=dist[current], for(i=0;i<=n;i++) do if(perm[i]==0) do new1=start+g[current][i].wt , if(new1<dist[i]) do dist[i]=new1; if(dist[i]<small) do small=dist[i] , temp=i ; current=temp, perm[current]=1, k++, display(current).

Step 5 – return(small).

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define member 1
#define nomember 0
#define infinity 999
typedef struct edge
{
   int wt;
}edge;
int n;
edge g[10][10];
void main()
{
   int src,dest;
   void get();
   int shrt(int,int);
   printf("\nOUTPUT\n");
   printf("\nPROGRAM FOR DIJKSTRA'S ALGORTHIM\n");
   printf("\nEnter the number of vertices: ");
   scanf("%d",&n);
   get();
   printf("\nEnter the sources: ");
   scanf("%d",&src);
   printf("\nEnter the destination: ");
   scanf("%d",&dest);
   printf("\nShortest path %d",shrt(src,dest));
   getch();
}
void get()
{
   int i,j,v1,v2;
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         printf("\nEnter the edges of v %d to v %d: ",i,j);
         scanf("%d",&g[i][j].wt);
      }
      printf("\n");
   }
}
int shrt(int src,int dest)
{
```

```
    int small,perm[10],dist[10],current,start,new1;
    int k=1,temp,i;
    for(i=1;i<=n;i++)
    {
       perm[i]=0;
       dist[i]=infinity;
    }
    perm[src]=1;
    dist[src]=0;
    current=src;
    while(current!=dest)
    {
       small=infinity;
       start=dist[current];
       for(i=1;i<=n;i++)
       {
          if(perm[i]==0)
          {
             new1=start+g[current][i].wt;
             if(new1<dist[i])
             dist[i]=new1;
             if(dist[i]<small)
             {
                small=dist[i];
                temp=i;
             }
          }
       }
       current=temp;
       perm[current]=1;
       k++;
       printf("%d\n",current);
    }
    return (small);
 }
```

OUTPUT:

```
OUTPUT

PROGRAM FOR DIJKSTRA'S ALGORTHIM

Enter the number of vertices: 4

Enter the edges of v 1 to v 1: 0
```

Enter the edges of v 1 to v 2: 11

Enter the edges of v 1 to v 3: 12

Enter the edges of v 1 to v 4: 5

Enter the edges of v 2 to v 1: 11

Enter the edges of v 2 to v 2: 0

Enter the edges of v 2 to v 3: 6

Enter the edges of v 2 to v 4: 12

Enter the edges of v 3 to v 1: 13

Enter the edges of v 3 to v 2: 6

Enter the edges of v 3 to v 3: 0

Enter the edges of v 3 to v 4: 7

Enter the edges of v 4 to v 1: 5

Enter the edges of v 4 to v 2: 12

Enter the edges of v 4 to v 3: 7

Enter the edges of v 4 to v 4: 0

Enter the sources: 1

Enter the destination: 4
4
Shortest path 5

RESULT:

Thus the C program to find the shortest path to all the other vertices by using Dijkstra's algorithm was written and output is verified successfully**.**

EX.NO : 12B	SHORTEST PATHS – FLOYD'S ALGORITHM
DATE :	

AIM:

To write a C program to find the shortest path from vertex to all the other vertices by using Floyd's algorithm.

ALGORITHM:

Step 1 – Start

Step 2 – Read the number of vertices

Step 3 – for(i=0;i<a;i++) do for(j=0;j<a;j++) do if(i==j) set x[i][j]=0 else read if there is an edge from i+1 vertex to j+1 vertex if(y==1) read weight of that edge and set x[i][j]=m else set x[i][j]=infinity.

Step 4 – for(k=0;k<a;k++) do for(i=0;i<a;i++) do if(x[i][j]>(x[i][k]+x[k][j])) set x[i][j]=x[i][k]+x[k][j].

Step 5 - for(i=0;i<a;i++) do for(j=0;j<a;j++) do display x[i][j].

Step 6 - Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define infinity 100000
void main()
{
   int i,j,k,a,x[50][50],y,m;
   printf("\t\t\t*****FLOYD'S ALGORITHM*****");
   printf("\nenter the number of vertices:");
   scanf("%d",&a);
   for(i=0;i<a;i++)
   {
      for(j=0;j<a;j++)
      {
         if(i==j)
         {
            x[i][j]=0;
         }
```

```
                else
                {
                    printf("Is there an edge from vertex %d to %d:",i+1,j+1);
                    scanf("%d",&y);
                    if(y==1)
                    {
                        printf("enter the weight:");
                        scanf("%d",&m);
                        x[i][j]=m;
                    }
                    else
                    {
                        x[i][j]=infinity;
                    }
                }
            }
        }
        for(k=0;k<a;k++)
        {
            for(i=0;i<a;i++)
            {
                for(j=0;j<a;j++)
                {
                    if(x[i][j]>(x[i][k]+x[k][j]))
                    {
                        x[i][j]=x[i][k]+x[k][j];
                    }
                }
            }
        }
        printf("\nmatrix representation of shortest path from each vertex to all the other vertex for the given directed graph:\n");
        for(i=0;i<a;i++)
        {
            for(j=0;j<a;j++)
            {
                printf("%d\t\t",x[i][j]);
            }
            printf("\n");
        }
    }
```

OUTPUT:

*****FLOYD'S ALGORITHM*****

enter the number of vertices:4
Is there an edge from vertex 1 to 2:0
Is there an edge from vertex 1 to 3:1
enter the weight:3
Is there an edge from vertex 1 to 4:0
Is there an edge from vertex 2 to 1:1
enter the weight:2
Is there an edge from vertex 2 to 3:0
Is there an edge from vertex 2 to 4:0
Is there an edge from vertex 3 to 1:0
Is there an edge from vertex 3 to 2:1
enter the weight:7
Is there an edge from vertex 3 to 4:1
enter the weight:1
Is there an edge from vertex 4 to 1:1
enter the weight:6
Is there an edge from vertex 4 to 2:0
Is there an edge from vertex 4 to 3:0

matrix representation of shortest path from each vertex to all the other vertex for the given directed graph:

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

RESULT:

Thus the C program to find the shortest path from vertex to all the other vertices by using Floyd's algorithm was written and output is verified successfully**.**

EX.NO. : 13A	SELECTION SORT
DATE:	

AIM:

To write a c program to sort a given element by using selection sort.

ALGORITHM:

Step 1:start

Step 2:Read size of list

Step 3:read the element of list by using for loop

Step 4:Display original list by using for loop

Step 5:for(i=0;i<=-2;i++)

min=1

for(j=0;j<=n-2;j++)

if(a[j]<a[min])

min=j

x=a[min]

a[min]=a[i]

a[i]=x

Step 6:Display the sorterd list by using for loop

Step 7:Stop

PROGRAM:

```
#include<stdio.h>
int i,j,a[50],n,min,temp;
int main()
{
   printf("\n\n\t\t***SELECTION SORT***");
   printf("\nEnter the no. of elements to be sort:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter the element:");
      scanf("%d",&a[i]);
```

```
    }
    printf("\nThe elements before sorting:");
     for(i=0;i<n;i++)
     {
         printf("\n%d ",a[i]);
     }
     for(i=0;i<=n-2;i++)
     {
        min=i;
        for(j=i+1;j<=n-1;j++)
        {
           if(a[j]<a[min])
           {
           min=j;
           }
        }
        temp=a[min];
        a[min]=a[i];
        a[i]=temp;
     }
     printf("\nThe elements after sorting:");
     for(i=0;i<n;i++)
     {
        printf("\n%d ",a[i]);
     }
}
```

OUTPUT:

```
            ***SELECTION SORT***
Enter the no. of elements to be sort:7
Enter the element:89
Enter the element:45
Enter the element:68
```

Enter the element:90

Enter the element:29

Enter the element:34

Enter the element:17

The elements before sorting:

89

45

68

90

29

34

17

The elements after sorting:

17

29

34

45

68

89

90

RESULT:

Thus the c program to sort a given list by using selection sort was written and output is verified successfully.

EX.NO. : 13B	BUBBLE SORT
DATE:	

AIM:

To write a c program to sort a given element by using bubble sort.

ALGORITHM:

Step 1:start

Step 2:Read size of list

Step 3:read the element of list by using for loop

Step 4:Display original list by using for loop

Step 5:for(i=0;i<=-2;i++)

for(j=0;j<=n-2-i;j++)

if(a[j+1]<a[j])

x=a[j]

a[j]=a[j+1]

a[j+1]=x

Step 6:Display the sorted list by using for loop

Step 7:Stop

PROGRAM:

```
#include<stdio.h>
int i,j,a[50],n,temp;
int main()
{
   printf("\n\n\t\t***BUBBLE SORT***");
   printf("\nEnter the no. of elements to be sort:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter the element:");
      scanf("%d",&a[i]);
   }
```

```
    printf("\nThe elements before sorting:");
    for(i=0;i<n;i++)
    {
        printf("\n%d ",a[i]);
    }
    for(i=0;i<=n-2;i++)
    {
       for(j=0;j<=n-2-i;j++)
       {
          if(a[j+1]<a[j])
          {
             temp=a[j];
             a[j]=a[j+1];
             a[j+1]=temp;
          }
       }
    }
    printf("\nThe elements after sorting:");
    for(i=0;i<n;i++)
    {
       printf("\n%d ",a[i]);
    }
}
```

OUTPUT:

```
            ***BUBBLE SORT***
Enter the no. of elements to be sort:7
Enter the element:89
Enter the element:45
Enter the element:68
Enter the element:90
Enter the element:29
Enter the element:44
Enter the element:17
```

The elements before sorting:

89

45

68

90

29

44

17

The elements after sorting:

17

29

44

45

68

89

90

RESULT:

Thus the c program to sort a given list by using bubble sort was written and output is verified successfully.

EX.NO. : 13C	INSERTION SORT
DATE:	

AIM:

To write a c program to sort a given element by using insertion sort.

ALGORITHM:

Step 1:start

Step 2:Read size of list

Step 3:read the element of list by using for loop

Step 4:Display original list by using for loop

Step 5:for(i=1;i<=n-1;i++)

v=a[i]

j=j-1

while (j>=0and a[j]>v)

a[j+1]=a[j]

j=j-1

a[j+1]=v

Step 6:Display the sorted list by using for loop

Step 7:Stop

PROGRAM:

```
#include<stdio.h>
int i,j,a[50],n,v;
int main()
{
   printf("\n\n\t\t***INSERTION SORT*");
   printf("\nEnter the no. of elements to be sort:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter the element:");
      scanf("%d",&a[i]);
   }
```

```
    printf("\nThe elements before sorting:");
     for(i=0;i<n;i++)
     {
         printf("\n%d ",a[i]);
     }
     for(i=0;i<=n-1;i++)
     {
        v=a[i];
        j=i-1;
        while(j>=0 && a[j]>v)
        {
          a[j+1]=a[j];
          j=j-1;
        }
        a[j+1]=v;
     }
     printf("\nThe elements after sorting:");
     for(i=0;i<n;i++)
     {
        printf("\n%d ",a[i]);
     }
}
```

OUTPUT:

```
            ***INSERTION SORT*
Enter the no. of elements to be sort:7
Enter the element:89
Enter the element:45
Enter the element:68
Enter the element:90
Enter the element:29
Enter the element:34
Enter the element:17
```

The elements before sorting:

89

45

68

90

29

34

17

The elements after sorting:

17

29

34

45

68

89

90

RESULT:

Thus the c program to sort a given list by using insertion sort was written and output is verified successfully.

EX.NO : 14A	QUICK SORT
DATE :	

AIM:

To write a C Program for implement quick sort.

ALGORITHM:

Step 1 - Start

Step 2 - Read the size of the list(a).

Step 3 - Read the element of list by using for loop.

Step 4 - Display the original list using for loop.

Step 5 - quicksort(a,0,n-1)

Step 6 – Display the sorted list using for loop.

Step 6 - Stop

QUICKSORT()

Step 1- Start

Step 2 - If (low<high)

p=partion(a,low,high)

quicksort(a,low,p-1)

quicksort(a,p+1,high)

Step 3 - Stop

PARTITION()

Step 1- Start

Step 2 - Set pivot=a[low], i=low+1, j=high

Step 3 - While(i<=j) do

While(a[i]<=pivot) do i++

While(a[j]>pivot) do j—

If (i<j) do swap a[i] and a[j]

Step 4 - Swap a[j] and a[low]

Step 5 - return j

Step 6 - Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int p,i,j,pivot,n,temp,a[50];
void quicksort(int[],int,int);
int partition(int[],int,int);
void quicksort(int a[50],int low,int high)
{
if(low<high)
 {
   p=partition(a,low,high);
   quicksort(a,low,p-1);
   quicksort(a,p+1,high);
  }
}
int partition(int a[50],int low,int high)
{
   pivot=a[low];
   i=low+1;
   j=high;
   while(i<=j)
   {
      while(a[i]<=pivot)
         i++;
      while(a[j]>pivot)
         j--;
```

```
            if(i<j)
            {
               temp=a[i];
               a[i]=a[j];
               a[j]=temp;
            }
         }
         temp=a[j];
         a[j]=a[low];
         a[low]=temp;
         return j;
      }
      void main()
      {
         printf("\t\t\t***QUICK SORTING***\n");
         printf("\nEnter the size of the list:");
         scanf("%d",&n);
         for(i=0;i<n;i++)
         {
            printf("\nEnter the element of list:");
            scanf("%d",&a[i]);
         }
         printf("\nORIGINAL LIST:\n");
         for(i=0;i<n;i++)
         {
            printf("%d\t",a[i]);
         }
         quicksort(a,0,n-1);
         printf("\nSORTED LIST:\n");
         for(i=0;i<n;i++)
         {
            printf("%d\t",a[i]);
         }
      }
```

OUTPUT:

```
                        ***QUICK SORTING***

      Enter the size of the list: 5

      Enter the element of list: 5

      Enter the element of list: 1

      Enter the element of list: 9
```

Enter the element of list: 3

Enter the element of list: 6

ORIGINAL LIST:
5 1 9 3 6
SORTED LIST:
1 3 5 6 9

RESULT:

Thus the C program to implement quick sort was written successfully and output is verified.

EX.NO : 14B	MERGE SORT
DATE :	

AIM:

To write a C Program for implement merge sort.

ALGORITHM:

Step 1 - Start

Step 2 - Read the size of the list(a).

Step 3 - Read the element of list by using for loop.

Step 4 - Display original list using for loop.

Step 5 - mergesort(a,0,n-1)

Step 6 – Display sorted list using for loop.

Step 6 - Stop

MERGESORT()

Step 1- Start

Step 2 - If (low<high)

mid=(low+high)/2

mergesort(a,low,mid)

mergesort(a,mid+1,high)

merge(a,low,mid,high)

Step 3 - Stop

MERGE()

Step 1- Start

Step 2 - Set i=low , j=mid+1, k=low

Step 3 - While(i<=mid && j<=high) do

If (a[i]<=a[j]) do

b[k]=a[i]

i++ ,k++

Else do

b[k]=a[j]

j++, k++

Step 4 - While(i<=mid)

b[k]=a[i]

i++ ,k++

Step 5 - While(j<=high)

b[k]=a[j]

j++ ,k++

Step 6 - Copy elements of list (b) to list (a) using for loop.

Step 7 - Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,n,a[50],b[50];
void merge(int[],int,int,int);
void mergesort(int[],int,int);
void mergesort(int a[50],int low,int high)
{
   int mid;
   if(low<high)
   {
      mid=(low+high)/2;
      mergesort(a,low,mid);
      mergesort(a,mid+1,high);
```

```
        merge(a,low,mid,high);
    }
}
void merge(int a[50],int low,int mid,int high)
{
   i=low;
   j=mid+1;
   k=low;
   while(i<=mid && j<=high)
   {
     if(a[i]<=a[j])
     {
       b[k]=a[i];
       i++;
       k++;
     }
     else
     {
       b[k]=a[j];
       j++;
       k++;
     }
   }
   while(i<=mid)
   {
      b[k]=a[i];
      i++;
      k++;
   }
   while(j<=high)
   {
      b[k]=a[j];
      j++;
      k++;
   }
   for(i=low;i<=high;i++)
   {
      a[i]=b[i];
   }
}
void main()
{
   printf("\t\t\t***MERGE SORTING***\n");
   printf("\nEnter the size of the list:");
```

```
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
       printf("\nEnter the element of list:");
       scanf("%d",&a[i]);
    }
    printf("\nORIGINAL LIST:\n");
    for(i=0;i<n;i++)
    {
       printf("%d\t",a[i]);
    }
    mergesort(a,0,n-1);
    printf("\nSORTED LIST:\n");
    for(i=0;i<n;i++)
    {
       printf("%d\t",b[i]);
    }
}
```

OUTPUT:

MERGE SORTING

Enter the size of the list: 5

Enter the element of list: 5

Enter the element of list: 9

Enter the element of list: 1

Enter the element of list: 7

Enter the element of list: 3

ORIGINAL LIST:
5 9 1 7 3
SORTED LIST:
1 3 5 7 9

RESULT:

Thus the C program to implement merge sort was written successfully and output is verified.

EX.NO : 15	HASHING APPLICATION
DATE :	

AIM:

To write a C program to implement a hashing techniques by using collision handling by Linear Probing.

ALGORITHM:

Step 1 – Start

Step 2 – Read the number to be insert.

Step 3 – Find the key by using create function.

Step 4 – Insert the number by using linear probing method.

Step 5 – Read if you need to continue it. If 'y', then repeat the step 2 to step 5 until it is not 'y'.

Step 6 – stop

CREATE()

Step 1 – Start

Step 2 – key=n%MAX

Step 3 - Return key

LINEAR_PROB()

Step 1 – Start

Step 2 – if(a[key]==-1) do a[key]=n

Step 3 - else set i=0 and while(i<MAX) do if(a[i]!=-1) do count++ and increment i until it gets fail and check if(count == MAX) do display(a) and return 1 and set i=(key+1)%MAX and while (1!=key) do if(a[i]==-1) do a[i]=n and break else i=(i+1)%MAX.

Step 4 –Stop

DISPLAY()

Step 1 – start

Step 2 – for(i=0;i<MAX;i++) do print(a[i]) until while condition is get fail.

Step 3 - stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void main()
{
        int a[MAX],n,key,i;
        char ans;
        int create(int);
        void linear_prob(int[],int,int),display(int[]);
        printf("\nCollision Handing by Linear Probing");
        for(i=0;i<MAX;i++)
        a[i]=-1;
        do
        {
    printf("\nEnter the number:");
    scanf("%d",&n);
    key=create(n);
    linear_prob(a,key,n);
    printf("\nDo you want to Continue?(Y/N)");
    ans=getch();
       }while(ans=='y' || ans=='Y');
        display(a);
        getch();
}
int create(int n)
{
        int key;
        key=n%MAX;

return key;
}
void linear_prob(int a[MAX],int key,int n)
{
   int i,count=0;
   void display(int a[]);
   if(a[key]==-1)
```

```
            a[key]=n;
        else
        {
            i=0;
            while(i<MAX)
            {
                if(a[i]!=-1)
                    count++;
                i++;
            }
            if(count==MAX)
            {
                printf("\nHash Table is full");
                display(a);
                getch();
                exit(1);
            }
            i=(key+1)%MAX;
            while(i!=key)
            {
            if(a[i]==-1)
            {
                a[i]=n;
                break;
            }
            i=(i+1)%MAX;
            }
        }
    }


    void display(int a[MAX])
    {
            int i;
            printf("\nThe Hash Table is\n");
            for(i=0;i<MAX;i++)
            printf("\n %d \t %d",i,a[i]);
    }
```

OUTPUT:

```
Collision Handing by Linear Probing
Enter the number:89
```

```
Do you want to Continue?(Y/N)
Enter the number:18

Do you want to Continue?(Y/N)
Enter the number:49

Do you want to Continue?(Y/N)
Enter the number:58

Do you want to Continue?(Y/N)
Enter the number:69

Do you want to Continue?(Y/N)
Enter the number:15

Do you want to Continue?(Y/N)
The Hash Table is

0    49
1    58
2    69
3    -1
4    -1
5    15
6    -1
7    -1
8    18
9    89
```

RESULT:

Thus the C program to implement a hashing techniques by using collision by linear probing was written and output is verified successfully**.**